

**Politechnika Warszawska**

W Y D Z I A Ł M A T E M A T Y K I  
I N A U K I N F O R M A C Y J N Y C H



# Praca dyplomowa inżynierska

na kierunku Informatyka

Interfejs użytkownika do manualnego obrysu struktur  
oraz wybranych anormalności w obrazach medycznych  
z wykorzystaniem tabletu graficznego

**Łukasz Garstecki**

Numer albumu 276857

**Tomasz Świerczewski**

Numer albumu 276915

Promotor

dr inż. Magdalena Jasionowska

WARSZAWA 2019

.....  
podpis promotora

.....  
podpisy autorów

## **Streszczenie**

Niniejszy dokument prezentuje pracę dyplomową inżynierską pod tytułem "Interfejs użytkownika do manualnego obrysu struktur oraz wybranych anormalności w obrazach medycznych z wykorzystaniem tabletu graficznego". Celem realizowanym przez autorów pracy jest stworzenie narzędzia informatycznego pozwalającego na manualne oraz półautomatyczne obrysowywanie obrazów medycznych. Zakres pracy obejmuje omówienie: omówienie standardu DICOM, aktualnego rynku aplikacji pozwalających na przeglądanie i wykonywanie obrysów na obrazach medycznych DICOM, architektury tworzonego systemu i eksperymentów przeprowadzonych w trakcie tworzenia narzędzia.

Zastosowana architektura umożliwia jednoczesne korzystanie z narzędzia przez wielu użytkowników. Opracowany na potrzeby generowania obrysów algorytm wykorzystuje operator Canny'ego oraz algorytm wyszukiwania najkrótszych ścieżek w grafie. Obliczane przez system statystyki dotyczące obrysowanych obszarów mogą posłużyć w przyszłości jako parametry wejściowe do metod automatycznego wykrywania podobnych do struktur, dla których obrysów wykonane.

**Słowa kluczowe:** interfejs użytkownika, tablet graficzny, obrazy medyczne, DICOM, obrys, statystyki danych obrazowych, system informatyczny, interfejs REST API, wykrywanie krawędzi, generowanie obrysów



## **Abstract**

The following document describes bachelor's thesis on "User interface for graphic-tablet interactions for contouring of structures and selected anomalies in medical images". The goal of authors' is to create a digital tool that allows to manually draw and semi-automatically generate contours on medical images. The topics discussed include: discussion of DICOM standard, analysis of state-of-the-art systems and tools used to viewing and creating contours on DICOM medical images, system's architecture and experiments conducted simultaneously with the creation of the tool.

The architecture used enables simultaneous use of the tool by many users. The algorithm developed for the purpose of generating contours uses Canny's Operator and the algorithm of searching the shortest path in the graph. The statistics calculated by the system regarding the areas limited by contours can be used in future as input parameters for methods of automatic detection of similar structures for which contours have been made.

**Keywords:** user interface, graphics tablet, medical images, DICOM, contour, statistics of image data, IT system, REST API interface, edge detection, contour generation



Warszawa, dnia .....

### Oświadczenie

Oświadczam, że moją część pracy inżynierskiej (zgodnie z podziałem zadań opisanym na wstępie) pod tytułem „Interfejs użytkownika do manualnego obrysu struktur oraz wybranych anormalności w obrazach medycznych z wykorzystaniem tabletu graficznego”, której promotorem jest dr inż. Magdalena Jasionowska, wykonałem samodzielnie, co poświadczam własnoręcznym podpisem.

.....



## **Spis treści**

# Wstęp

W niniejszym rozdziale przedstawiono cel pracy inżynierskiej wraz z motywacją, która kierowała autorami przy wyborze właśnie tego tematu.

## Cel pracy

Celem niniejszej pracy inżynierskiej jest stworzenie systemu czy też narzędzia informatycznego umożliwiającego tworzenie obrysów plików medycznych. Użytkownik może otwierać pliki medyczne DICOM i dokonywać obrysów manualnych lub półautomatycznych. W przypadku obrysów manualnych użytkownik może tworzyć obrys w obrysach w celu zaznaczania anomalii. Użytkownik ma mieć zapewnioną możliwość korzystania z tabletu graficznego w celu wygodniejszego rysowania obrysów.

Ma zostać stworzony algorytm półautomatyczny do tworzenia obrysów półautomatycznych. Jest to metoda półautomatyczna, ponieważ jest wspomagana przez człowieka, który wybiera punkty na obrazie medycznym, które algorytm będzie interpolował. Do tego algorytmu półautomatycznego zostanie zastosowane rozwiązanie inspirowane wieloma algorytmami przetwarzania obrazów medycznych, czyli operator Canny'ego i późniejsze przetwarzanie informacji dzięki zastosowania grafów. Zostanie przeprowadzona walidacja, czy ta metoda sprawdza się w celu usprawnienia rysowania obrysów manualnych.

Zaproponowane przez autorów rozwiązanie opiera się na aplikacji przeglądarkowej i serwera przechowującego obrys. To rozwiązanie powinno być wygodniejsze do użycia w sieciach lokalnych, gdzie użytkownik lub użytkownicy na dowolnej stacji roboczej mogą wykonywać obrys, a one mają się zapisywać w wspólnym miejscu.

## Motywacja

W czasach gwałtownego rozwoju sztucznej inteligencji jest ona stosowana prawie do każdej dziedziny z szeroko pojętej nauki. Jedną z takich dziedzin jest medycyna. Próba odnajdywania

w wczesnym stadium różnych niebezpiecznych zmian może ocalić wiele ludzkich istnień, więc obecnie na całym świecie trwają prace nad automatycznymi sposobami wykrywania takich zmian na obrazach medycznych.

W celu odnajdywania obrazów medycznych potrzebna jest wiedza czego szukać, jak odnajdywać narządy i zmiany wewnętrz nich. Z tego powodu potrzeba stworzyć narzędzie, które w wygodny sposób pozwoli nam klasyfikować obrysów wybranych narządów i anormalności medycznych. Na podstawie tych zebranych danych można w dalszej przyszłości tworzyć zaawansowane narzędzia do automatycznej klasyfikacji.

# **1. Wprowadzenie**

Rozdział ten stanowi wstęp do medycyny pod kątem przetwarzania obrazów medycznych. Wyjaśniono w nim, co to jest plik DICOM i jakie badania są reprezentowane tym formatem pliku. Ponadto opisano podział prac w ramach tego systemu, czy też narzędzia informatycznego.

## **1.1. Zagadnienia medyczne związane z aplikacją**

DICOM (ang. Digital Imaging and Communications in Medicine) [?] to norma opracowana dla potrzeb wymiany i interpretacji danych medycznych, które są najczęściej obrazami medycznymi. W postaci pliku DICOM mogą być zapisywane obrazy tomografii komputerowej, obrazowania metodą rezonansu magnetycznego, cyfrowej angiografii subtrakcyjnej, zdjęcia rentgenowskie, mammografia. Do wad tego standardu należy brak standardu, czyli czasem nie można wyświetlić pliku na sprzęcie innego producenta i należy ręcznie zmieniać parametry. Ten fakt generuje problemy z ewentualnym zapisem obrysu do pliku DICOM, ponieważ w takim przypadku obecnie tylko stworzony w ramach tej pracy system byłby w stanie poprawnie odczytać taki plik DICOM. Ponadto ten standard ma dosyć wysoki próg wejścia, czyli wiedzy potrzebnej na płynne użytkowanie jego, ponieważ może zawierać do 12 tysięcy wskaźników, zwanych tagami.

Do przeglądania plików w formacie standardu DICOM istnieje różne oprogramowanie. Jak każda aplikacje mają swoje wady, ponieważ wiele z nich jest ukierunkowana na jeden typ badania np. do badania tomografii komputerowej, inne nie wspierają plików wygenerowanych na części urządzeń.

Podsumowując, mimo, że DICOM jest standardem pliku obrazu medycznego, to pojawiają się problemy z jego użytkowaniem.

## **1.2. Podział prac**

Do obowiązków Łukasza Garsteckiego należeć będą:

- z projektowaniem interfejsu użytkownika w aplikacji przeglądarkowej,
- wczytywanie plików medycznych do aplikacji przeglądarkowej,
- stworzenie modułu anonimizacji danych pacjenta i zmodyfikowaniu plików medycznych,
- stworzenie narzędzia do rysowania obrysu manualnego,
- stworzenie narzędzia do wybierania punktów do obrysu półautomatycznego,
- stworzenie narzędzia do wizualizacji stworzonych obrysów przez użytkownika.

Zadaniami Tomasza Świerczewskiego będą:

- stworzenie części serwerowej systemu wraz z bazą danych,
- opracowanie algorytmu do generowania obrysów półautomatycznych,
- stworzenie API, które będzie udostępniało obrysy,
- stworzenie modułu statystyk,
- przygotowanie i przeprowadzenie eksperymentów.

## **2. Stan wiedzy**

W poniższym rozdziale zostały przedstawione istniejące na rynku aplikacje oferujące podobne funkcjonalności do wymagań postawionych przed autorskim systemem do obrysów na obrazach DICOM. Przedstawiono również proponowane rozwiązanie postawionych przed systemem wymagań.

### **2.1. Przegląd istniejących rozwiązań**

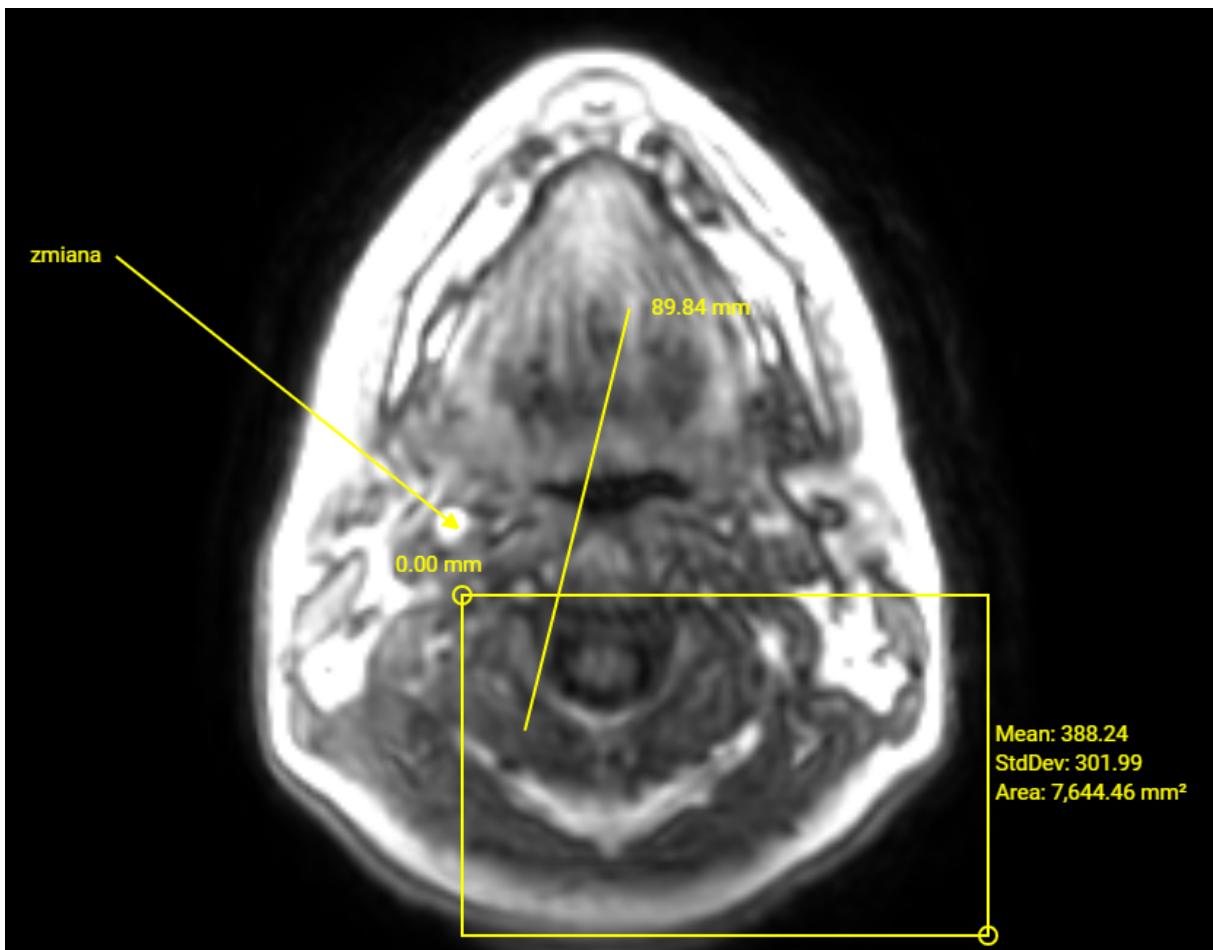
#### **2.1.1. Cornerstone**

Cornerstone Core [?] to biblioteka, która umożliwia wyświetlanie obrazów medycznych z wykorzystaniem elementu canvas z języka HTML5. Biblioteka udostępnia interfejs do wyświetlania obrazów medycznych pozwalający na zarządzanie wyświetlaniem zdjęcia. Podstawowe funkcjonalności obsługiwane przez bibliotekę to:

- przybliżanie i oddalanie obrazu,
- obrót obrazu,
- przesuwanie obrazu w wyświetlanym komponencie,
- zmiana jasności wyświetlanego obrazu,
- mapowanie kolorów,
- interpolacja pikseli w obrazie (dla obrazów o niskiej rozdzielczości).

Ponadto twórcy biblioteki Cornerstone Core stworzyli bibliotekę Cornerstone Tools, która korzysta z biblioteki Cornerstone Core i umożliwia wiele funkcjonalności potrzebnych lekarzom do analizy badań pacjentów. Poza funkcjonalnościami Cornerstone Core umożliwia także:

- Mierzenie odległości w linii prostej na obrazie z podaniem rzeczywistych wartości,
- Oznaczanie obszarów przy pomocy prostokątów oraz elips,
- Oznaczanie niewielkich zmian w postaci małego okręgu,
- Mierzenie kątów na podstawie 3 podanych przez użytkownika punktów.



Rysunek 2.1: Przykład użycia aplikacji OHIF Viewer

Przykładowym projektem korzystającym z bibliotek Cornerstone jest OHIF Viewer. Aplikacja pozwala na wykorzystanie większości możliwości udostępnianych przez biblioteki Cornerstone. Przykładowe użycie aplikacji zostało przedstawione na rysunku ??.

### 2.1.2. DICOM Web Viewer (DWV)

Aplikacja DICOM Web Viewer (DWV) [?] jest również przykładem aplikacji przeglądarkowej służącej do przeglądania obrazów DICOM z wygodnym interfejsem automatycznego i półautomatycznego obrysowywania interesujących użytkownika obszarów zaznaczonych na obrazie. Przykładowy obrys wykonany przy użyciu aplikacji DWV przedstawiono na rysunku ??.

Poza tym aplikacja udostępnia:

- przybliżanie i oddalanie obrazu,
- przesuwanie obrazu w wyświetlanym komponencie,
- zmiana jasności wyświetlanego obrazu,
- mierzenie odległości w linii prostej na obrazie z podaniem rzeczywistych wartości,

## 2.2. PROPONOWANE ROZWIĄZANIE

- oznaczanie obszarów przy pomocy prostokątów oraz elips.

Aplikacja nie pozwala na wykonywanie manualnych obrysów poprzez samodzielne poruszanie kursorem po obrazie, więc nie spełnia wymagań postawionych przed realizowanym narzędziem.

### 2.1.3. Orthanc

Orthanc to serwer z bazą plików DICOM, który umożliwia łatwe przechowywanie, zarządzanie oraz dostęp do plików medycznych DICOM. Ponadto Orthanc udostępnia REST API, które umożliwia przeglądanie wgranych na serwer plików DICOM podzielonych względem pacjentów, badań i serii.

Serwer Orthanc udostępnia również prosty interfejs webowy, który pozwala na wgrywanie nowych plików (interfejs wgrywania plików przedstawiono na Rysunku ??), przeglądanie zapisanych plików — w szczególności tagów (rysunek ??) oraz podglądu obrazu (Rysunek ??). Interfejs webowy nie zapewnia żadnej metody rysowania na przeglądany obrazie. Pozwala natomiast na przełączanie obrazu na kolejny przez kliknięcie lewym przyciskiem myszy w prawą część podglądu obrazu.

### 2.1.4. Podsumowanie

Istniejące rozwiązania nie zapewniają wygodnego interfejsu użytkownikowi, który dysponuje tabletem graficznym i chciałby obrysowywać interesujące go części obrazów medycznych odręcznie. Ponadto żadne z narzędzi nie przechowuje narysowanych manualnie ani wygenerowanych automatycznie obrysów. Rozwiązanie przedstawione w tej pracy udostępnia takie możliwości.

## 2.2. Proponowane rozwiązanie

W rozwiążaniu zdecydowano się skorzystać z serwera Orthanc jako bazy przechowującej pliki DICOM. Przy decyzji miały znaczenie przede wszystkim to, że jest to popularne narzędzie. Ponadto udostępniane przez API serwera funkcjonalności są wystarczające dla realizowanego systemu.

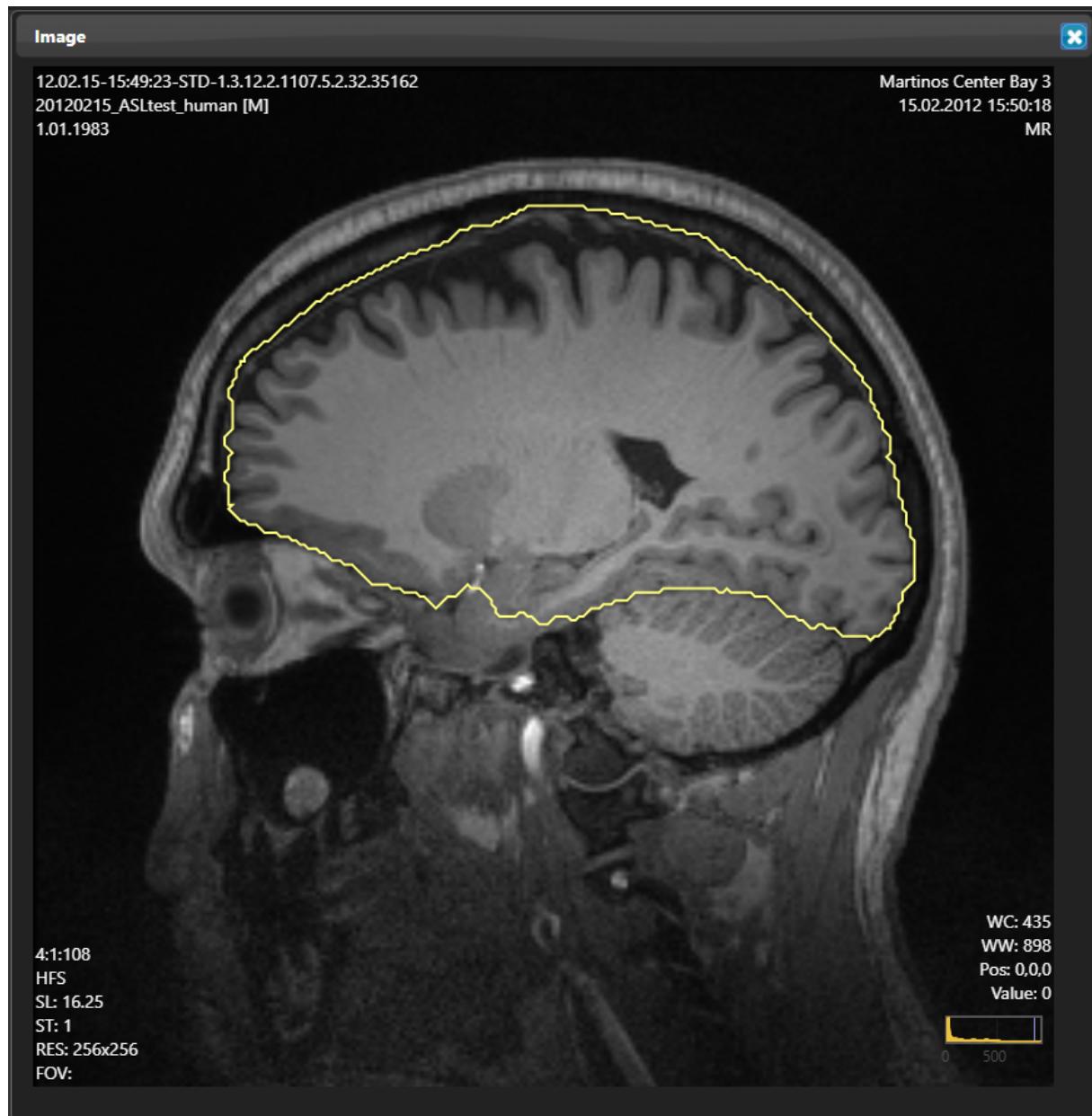
Poza serwerem Orthanc stworzono serwer w technologii .NET Core [?] wspomagający interfejs użytkownika, który pozwala na przeprowadzanie czasochłonnych aplikacji poza interfejsem użytkownika. Wybrano tę technologię ze względu na to, że jest w ciągłym rozwoju i nie ustępuje jakością w stosunku do innych rozwiązań [?]. Podstawowymi zadaniami tego serwera było przechowywanie wykonanych przez użytkownika obrysów manualnych, generowanie i przechowywanie obrysów półautomatycznych oraz obliczanie statystyk związanych z zapisanymi obrysami.

Jako interfejs użytkownika wykorzystano bibliotekę ReactJS [?] w połączeniu z biblioteką Redux. Zdecydowano się na tę technologię ze względu na dużą popularność i elastyczność [?]. Wybranie tej technologii pozwoliło na stworzenie aplikacji przeglądarkowej pozwalającej na generowanie obrysów manualnych przy użyciu myszy lub tabletu graficznego, wybieranie punktów, z których generowane będą obrys półautomatyczne, oglądanie wcześniej wygenerowanych i zapisanych obrysów oraz wyświetlanie statystyk dotyczących wykonanego obrysu. Interfejs inspirowany jest podglądem obrazu w aplikacji DWV, ale funkcjonalność automatycznego obrysu została zmieniona — obrys nie jest generowany na bieżąco, lecz na życzenie użytkownika wybrane przez niego punkty wysyłane są do serwera i wyświetlany jest wynik półautomatycznego dopasowania obrysów.

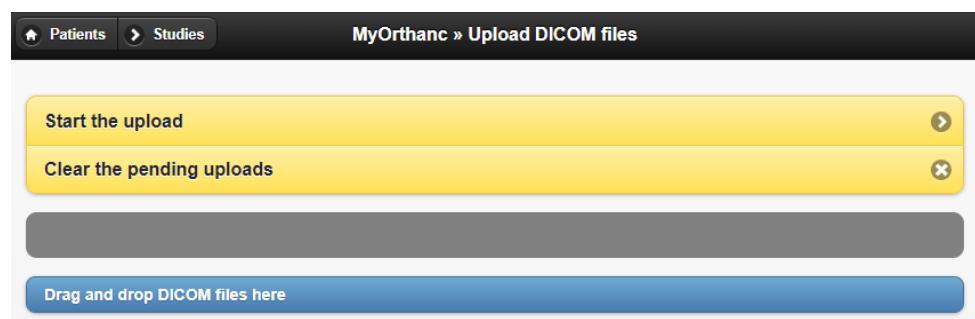


Rysunek 2.2: Podgląd obrazu DICOM przez interfejs przeglądarkowy Orthanc

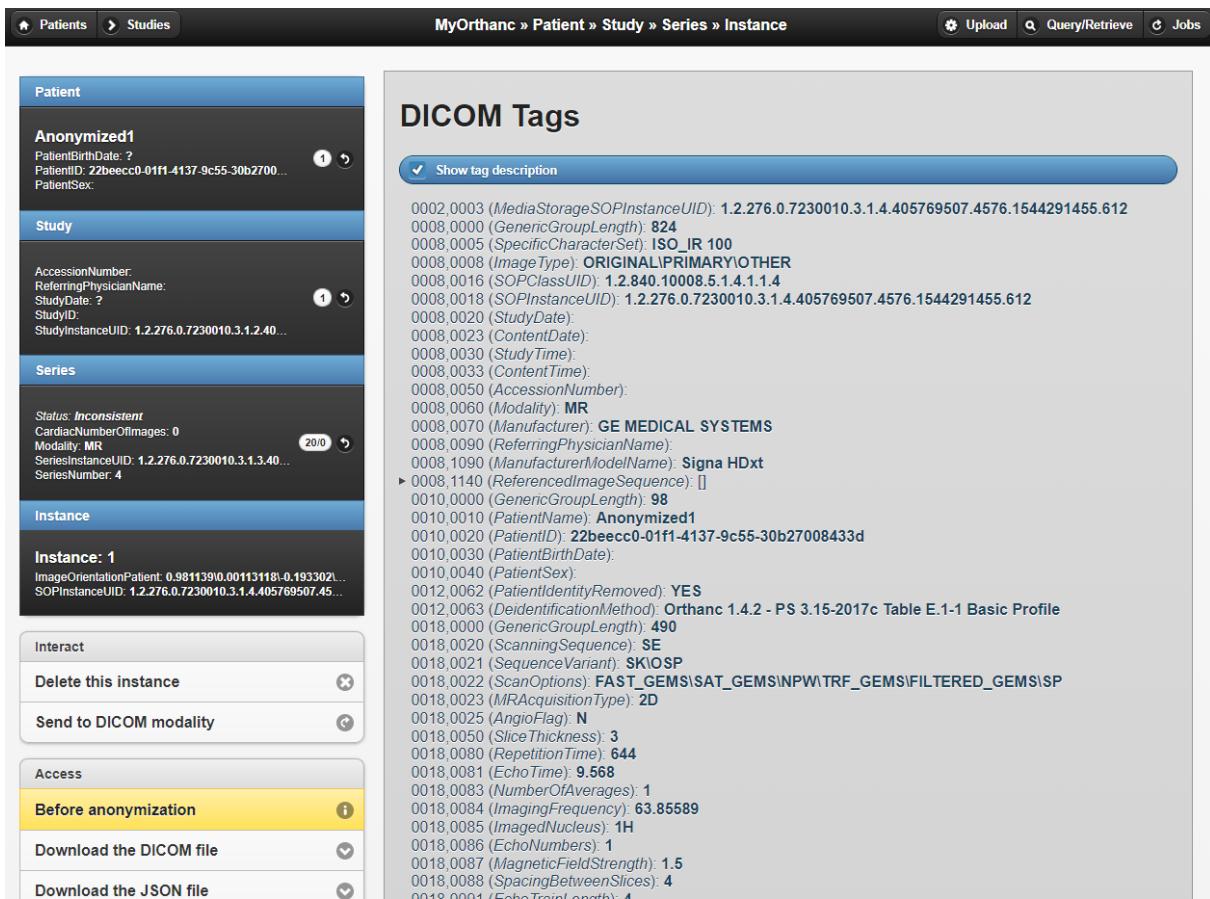
## 2.2. PROPONOWANE ROZWIĄZANIE



Rysunek 2.3: Przykład użycia aplikacji DWV - obrrys półautomatyczny (Livewire)



Rysunek 2.4: Interfejs wgrywania plików DICOM do serwera Orthanc



Rysunek 2.5: Podgląd tagów pliku DICOM przez interfejs przeglądarkowy Orthanc

### **3. Opis autorskiego narzędzia informatycznego**

W poniższym rozdziale zawarto dokumentację techniczną i biznesową tworzonego systemu. Przedstawiono w szczególności: wymagania funkcjonalne i niefunkcjonalne, architekturę, zastosowane metody półautomatycznego obrysu oraz metody obliczania statystyk obrysów.

#### **3.1. Specyfikacja wymagań**

Przed stworzeniem systemu zostały zgromadzone dane dotyczące wymagań stawianych przed tworzonym systemem. Zostały one przedstawione w postaci zgodnej z wymaganiami stawianymi w inżynierii oprogramowania w rozdziałach 3.1.1. - 3.1.3. Wymagania były gromadzone na podstawie wywiadów przeprowadzonych z opiekunem naukowym tej pracy dyplomowej, z lekarzami i ze studentami Warszawskiego Uniwersytetu Medycznego.

##### **3.1.1. Opis biznesowy**

Celem projektu jest stworzenie interfejsu przyjaznego użytkownikowi, który umożliwi przeglądanie plików DICOM, a także przeprowadzanie na tych plikach obrysów. Prace obejmują stworzenie aplikacji webowej, która udostępnii użytkownikowi interfejs komunikujący się z bazą danych Orthanc oraz serwera odpowiedzialnego za przechowywanie wygenerowanych przez użytkownika obrysów oraz wyznaczanie obrysów półautomatycznych.

Do podstawowych funkcjonalności systemu zaliczają się:

- generowanie obrysu manualnego.
- generowanie obrysu półautomatycznego na podstawie punktów wybranych przez użytkownika.
- zapisywanie wygenerowanych obrysów.
- anonimizacja<sup>1</sup> danych zapisanych w strukturze pliku DICOM.

---

<sup>1</sup>Anonimizacja (ang. anonymization) — operacja mająca na celu usunięcie z danych informacji o pacjentach, które pozwolilyby na identyfikację danych z tożsamością pacjenta. Są to między innymi: imiona, nazwisko, pesel. Inne tłumaczenia słowa anonymization — utajnianie, usuwanie danych niejawnych. Z uwagi na fakt, że te

#### 3.1.2. Wymagania funkcjonalne

Nieodłącznym elementem inżynierii oprogramowania przy próbie dokumentacji określonego produktu lub usługi, są wymagania funkcjonalne i niefunkcjonalne. Te pierwsze opisują funkcje i możliwości, które system powinien realizować. Zostały przygotowane wymagania funkcjonalne dla tworzonego systemu. Zostały one przedstawione poniżej w postaci historii użytkownika (ang. user stories), czyli czynności jakie może chcieć wykonać użytkownik tego systemu.

##### 1. Jako użytkownik chcę wczytać obraz DICOM.

Użytkownik może wybrać obraz w menu bocznym, w którym ma możliwość wyboru pacjenta, badania oraz serii. Wybranie serii skutkuje wyświetleniem pierwszego obrazu DICOM z tej serii.

##### 2. Jako użytkownik chcę zmienić obraz w serii przy użyciu rolki myszy.

Po umieszczenie kurSORA na obrazie przewijanie rolką myszy do góry powoduje zmianę wyświetlanego obrazu na kolejny obraz w serii. Gdy przewijamy rolką myszy do góry na ostatnim obrazie w serii wyświetlany obraz nie zmienia się. Analogicznie przewijanie rolką myszy w dół powoduje zmianę wyświetlanego obrazu na poprzedni obraz w serii, a przewijanie w dół rolką myszy na pierwszym obrazie w serii nie powoduje zmiany obrazu.

##### 3. Jako użytkownik chcę wykonać obrys przy użyciu tabletu graficznego.

Po umieszczenie kurSORA na obrazie sterowanym przez tablet graficzny, użytkownik prowadzi kurSOR po obrazie wykonując obrys bez odrywania końcówki rysaka od podkładki. Jeżeli użytkownik nie zakończy obrysu dokładnie w punkcie, w którym go rozpoczął, obrys powinien zakończyć się linią prostą, łączącą punkt końcowy z punktem początkowym.

##### 4. Jako użytkownik chcę wygenerować obrys na podstawie wybranych punktów.

Po umieszczenie kurSORA na obrazie użytkownik może wybierać punkty, na podstawie których zostanie wygenerowany obrys, poprzez wcisnięcie lewego przycisku myszy w miejscach, w których chce, aby znalazły się punkty. Użytkownik może zobaczyć efekt wygenerowanego przez system obrysu.

##### 5. Jako użytkownik chcę edytować listę punktów, z której wygenerowany zostanie obrys.

Użytkownik może usunąć wcześniej wybrany punkt po umieszczając nad nim kurSOR i wcisnięciu lewego przycisku myszy. Użytkownik może dodać nowy punkt do listy punktów poprzez wcisnięcie lewego przycisku myszy w miejscu, w którym chce wstawić punkt.

---

tłumaczenia nie oddają dobrze kontekstu, zastosowano kalkę językową.

### **3.1. SPECYFIKACJA WYMAGAŃ**

#### **6. Jako użytkownik chcę wybrać kolor obrysu.**

Użytkownik wybiera kolor z palety kolorów lub może zdefiniować własny kolor poprzez podanie kodu RGB koloru, który chce wybrać.

#### **7. Jako użytkownik chcę zapisać obrys.**

Po wykonaniu obrysu manualnego lub wybraniu listy punktów do wygenerowania obrysu półautomatycznego, użytkownik wybiera nazwę obrysu i zapisuje obrys w systemie.

#### **8. Jako użytkownik chcę obejrzeć zapisany obrys.**

Użytkownik wybiera z listy obrysów zapisany obrys i przegląda obrys naniesiony na obraz, na którym został wykonany.

#### **9. Jako użytkownik chcę zobaczyć statystyki dotyczące obrysu.**

Użytkownik wybiera z listy obrysów zapisany obrys i przegląda statystyki obliczone na podstawie zapisanego obrysu. Do statystyk zalicza się obwód obrysu, pole obrysu, histogram obrazu na obszarze obrysu oraz liczba pikseli wewnętrz obrysu.

#### **10. Jako użytkownik chcę zobaczyć jednocześnie dowolną liczbę zapisanych w systemie obrysów na jednym obrazie DICOM.**

Użytkownik wybiera poprzez kliknięcie lewym przyciskiem myszy na nazwie obrysu znajdującej się na liście obrysów. Wybrane obrysy wyświetlane są jednocześnie na przeglądany przez użytkownika zdjęciu. Użytkownik może wyłączyć podgląd wcześniej wybranego obrysu poprzez ponowne wcisnięcie lewego przycisku myszy na nazwie obrysu na liście po prawej stronie. Na zdjęciu wyświetlane są jedynie obrysy wykonane na tym obrazie.

#### **11. Jako użytkownik chcę zanomizować dane pacjenta zawarte w pliku DICOM.**

Użytkownik może zanomizować pacjenta, gdy przegląda jego obraz. Użytkownik może anonimizować imię i nazwisko pacjenta, datę urodzenia pacjenta oraz płeć pacjenta poprzez nadanie nowych wartości lub poprzez usunięcie poprzedniej wartości i pozostawienie pustych pól w formularzu.

#### **3.1.3. Wymagania niefunkcjonalne**

Drugim rodzajem wymagań są wymagania niefunkcjonalne. Opisują one kryteria osądzenia działania systemu pod kątem jakościowym. Założone wymagania niefunkcjonalne zostały przedstawione w Tabeli ??.

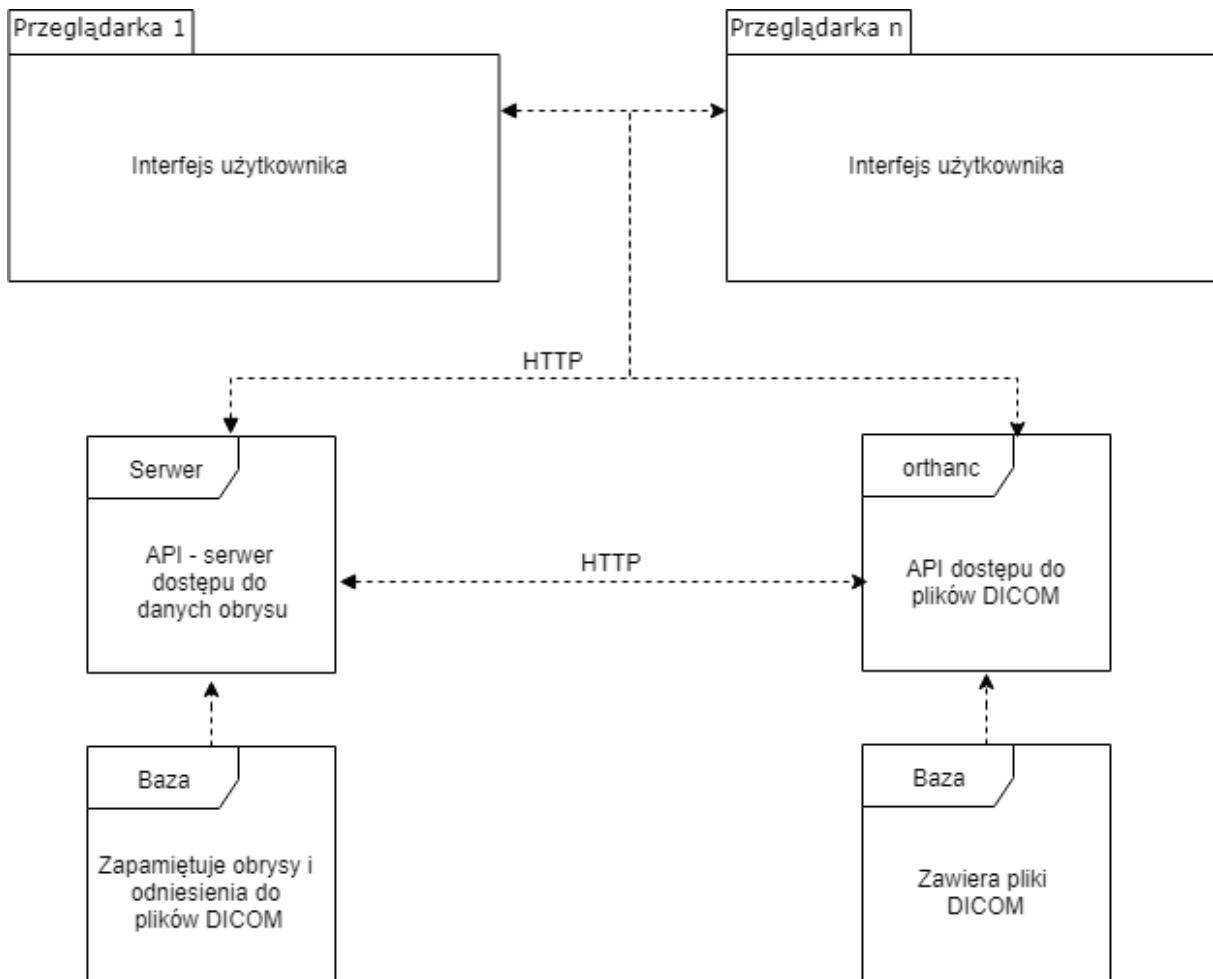
### 3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

Tablica 3.1: Spis wymagań niefunkcjonalnych

| Obszar wymagań                     | Opis   |
|------------------------------------|--|
| Użyteczność<br>(ang. Usability)    | Każda funkcjonalność aplikacji dostępna dla użytkownika musi mieścić się na pojedynczym ekranie przy rozdzielcości 1920x1080 i czcionce nie mniejszej niż 12pt.  |
|                                    | Aplikacja powinna udostępniać pobranie zapisanych obrysów przy użyciu serwisu REST.  |
| Niezawodność<br>(ang. Reliability) | Aplikacja ma być dostępna 24h w ciągu doby. Dopuszczalny jest brak działania aplikacji w dowolnym momencie przez okres nie dłuższy niż przez 12h. Po przerwie w działaniu aplikacja musi być dostępna przez kolejne 24h bez utrudnień. |
| Wydajność<br>(ang. Performance)    | Aplikacja powinna pobierać dane zewnętrzne w postaci pliku DICOM (około 20MB) nie dłużej niż 5 sekund.   |
|                                    | Aplikacja powinna generować obrys półautomatyczny i zapisywać obrys do systemu w czasie nie dłuższym niż 30 sekund.  |
|                                    | Aplikacja powinna reagować na działanie użytkownika (z wyłączeniem generowania obrysu półautomatycznego i zapisu obrysu do systemu) w czasie nie dłuższym niż 1 sekunda.   |

### 3.2. Architektura rozwiązania

Na Rysunku ?? przedstawiono architekturę autorskiego systemu.



Rysunek 3.1: Diiagram UML przedstawiający architekturę autorskiego systemu

Architektura pozwala na połączenie z serwerem plików DICOM (Orthanc) oraz serwerem obrysów dowolnej liczby klientów korzystających z przeglądarki. Algorytmy zapisu obrysów, generowania obrysów, obliczania statystyk oraz anonimizacji plików są od siebie niezależne i mogą zostać zrównoleglane — ograniczeniem jest tutaj moc obliczeniowa maszyny na której uruchomiony jest serwer. Niemożliwe jest zwiększenie wydajności poprzez zwiększenie liczby instancji serwera. Dopuszczalne jest takie rozwiązanie, gdyż planowana liczba użytkowników jednocześnie korzystających z aplikacji to nie więcej niż 20 osób. Możliwości zwiększenia skalowalności w kontekście obliczeniowej zostało opisane w rozdziale 5.2.

#### 3.2.1. Interfejs użytkownika

Kluczowym elementem stworzonego systemu jest interfejs użytkownika napisany w języku skryptowym JavaScript przy użyciu biblioteki ReactJS. Udostępnia ona interfejs nawigujący po zapisanych w serwerze Orthanc plikach DICOM. Ponadto dla wyświetlnego obrazu udostępnia szczegóły dotyczące obrazu, badania i pacjenta. Umożliwia także wykonanie obrysu manualnego zrealizowanego przy użyciu biblioteki react-canvas-draw. W aplikacji zawarto również autorski interfejs do wyboru punktów do algorytmu półautomatycznego, wykonany z użyciem elementu canvas [?]. Podgląd zapisanych w systemie obrysów wraz z wyliczonymi dla nich statystykami jest możliwy z poziomu tworzenia obrysów oraz w osobnej zakładce, gdzie udostępniono widok, w którym użytkownik może oglądać jednocześnie wybrane przez siebie obrysów z listy obrysów na jednym obrazie.

#### 3.2.2. Serwer obrazów Orthanc

Wyświetlane w interfejsie użytkownika obrazy DICOM oraz związane z nimi informacje przechowywane są przez serwer DICOM. Dostęp do zapisanych na serwerze danych jest możliwy poprzez udostępniane przez serwer API. Interfejs użytkownika wykorzystuje API poprzez protokoł HTTP.

API serwera udostępnia dane obrazów posegregowanych względem kolejno: pacjentów, badań oraz serii. Wykorzystano je w interfejsie użytkownika do nawigacji pomiędzy obrazami. Dodatkowo dla każdego obrazu serwer udostępnia dane obrazowe, umożliwiając wyświetlenie obrazu w interfejsie użytkownika. Ponadto API udostępnia wszystkie tagi obrazu DICOM, co umożliwia wyświetlanie interfejsowi użytkownika informacji w nich zawartych.

#### 3.2.3. Serwer obrysów i obliczeń

Kolejny kluczowy element stworzonego systemu to serwer obrysów i obliczeń. Udostępnia on API z obrysami, które zostało wykorzystane przez interfejs użytkownika. API zostało stworzone przy pomocy ASP.NET Core 2.2 [?] przy wykorzystaniu framework'a .NET Core 2.2 [?]. API oferuje 4 podstawowe operacje CRUD - utwórz, odczytaj, aktualizuj i usuń. W celu pobrania zdjęć medycznych serwer z API łączy się do bazy danych Orthanc. Informacje o obrysach, takie jak ID obrysu, tag, ID obrazu medycznego i typ obrysu (półautomatyczny lub manualny) są przechowywane w bazie danych Microsoft SQL Server. Pozostałe informacje, takie jak lista pikseli należących do obrysu, lista punktów w przypadku obrysu półautomatycznego i statystyki są przechowywane w plikach CSV. Te pliki są zapisywane w katalogu roboczym obok miejsca przechowywania plików źródłowych dla API.

### 3.3. MODUŁ OBRAZÓW MANUALNYCH

Poza serwerem Orthanc interfejs korzysta także z serwera przechowującego obrysów, a także wykonującego obliczenia związane z generowaniem obrysów. Serwer ten jest dostępny dla interfejsu użytkownika poprzez API dostępne przez protokół HTTP. Pozwala ono na pobieranie zapisanych obrysów, zapisywanie obrysów manualnych i generowanie podglądu i zapis obrysu półautomatycznego.

#### 3.3. Moduł obrazów manualnych

Obrysów manualnych generowanych przez użytkowników zapisywane są przez serwer obrysów i obliczeń w postaci plików CSV na dysku serwera. W momencie zapisu obliczane są również statystyki dotyczące obrysowanej części obrazu.

W związku z tym, że obrazy wyświetlane w interfejsie są w innej rozdzielczości niż rzeczywisty rozmiar obrazu, obrys przed zapisaniem zostają przeskalowane do faktycznych wymiarów obrazu. W przypadku obrazów o małych rozdzielczościach może spowodować to utratę dokładności obrysu, a w przypadku obrysów bardzo małych obszarów zdegradowanie obrysu do jednego piksela. Tego typu straty mogą wpływać na poprawność obliczanych statystyk jak również błędne wyświetlanie zapisanych obrysów w interfejsie użytkownika.

#### 3.4. Opracowany algorytm półautomatyczny

Opracowany algorytm półautomatyczny służy do wykrywania krawędzi na obrazie medycznym. Jest algorymem półautomatycznym, ponieważ jest wspomagany przez człowieka — użytkownika, który wybiera punkty na wyświetlanym obrazie. Te punkty są interpolowane przez algorytm półautomatyczny, zwany dalej algorytmem.

Jako dane wejściowe do algorytmu uzyskujemy następujące informacje:

- identyfikator obrazu medycznego, na którym był wykonywany obrys.
- lista punktów wybranych przez użytkownika. Punkty te zostały wcześniej przeskalowane ze współrzędnych w aplikacji internetowej (aplikacji webowej, ang. web application) na współrzędne odpowiadające rozdzielczości obrazu medycznego.

Algorytm można podzielić na kilka ważnych etapów:

- wykrycie krawędzi na bitmapie,
- stworzenie grafu z bitmapy,

### 3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

- zapewnienie spójności grafu,
- wyszukanie najkrótszych ścieżek w grafie.

Poniżej zostaną przedstawione dokładne rozwiązania dla każdego z tych kroków. Przed rozpoczęciem przetwarzania jest pobierany obraz medyczny o danym wcześniej identyfikatorze z serwera Orthanc. Jest on podstawą do dalszej pracy algorytmu.

#### 3.4.1. Wykrycie krawędzi na bitmapie

Często na obrazach medycznych różnice w charakterystyce poziomów szarości pikseli reprezentujących interesujące nas obiekty są małe, nie są podane dodatkowe informacje o naturze obrazu. Problem opracowania uniwersalnego algorytmu wykrywania krawędzi jest problemem trudnym. Dla wielu algorytmów można znaleźć takie przykłady, że te algorytmy nie wyznaczają poprawnie krawędzi. Te algorytmy muszą spełniać szereg wymagań, które stwierdzają poprawność danego algorytmu, czy też operatora morfologicznego. Zgodnie z Cytowski J. [?] „dobry detektor krawędzi powinien spełniać następujące warunki:

- niskie prawdopodobieństwo zaznaczenia punktów nienależących do krawędzi oraz niskie prawdopodobieństwo niezaznaczenia punktów należących do krawędzi,
- zaznaczone punkty krawędzi powinny być możliwie blisko jej osi,
- wyłącznie jedna odpowiedź na pojedynczy punkt krawędzi.”

Został użyty operator Canny'ego, bo jest on powszechnym i dobrze sprawdzonym rozwiązaniem do wykrywania krawędzi. Jak napisał autor [?] „Operator ten (Canny'ego) jest bardzo popularny, chętnie wykorzystywany i adoptowany do wielu zastosowań. (...) Stał on się również standardem często używanym do porównań innych metod wykrywania krawędzi.” Zgodnie z rysunkiem 4.25 „Porównanie operatorów wykrywania krawędzi” w [?] najlepiej wykrywał główne narządy, takie jak wątroba czy też trzustka. Z wyżej wymienionych powodów został on wykorzystany w tym algorytmie.

Operator Canny'ego [?] składa się z 3 zasadniczych kroków:

##### 1. Określenie wartości i kąta gradientu.

W tym celu został wykorzystany operator gradientu, a estymatorem gradientu w funkcji dyskretnej, jaką jest obraz, zastosowano maskę, czy też operator Sobela. Wykorzystując go uzyskano dla każdego piksela wielkość oraz kierunek gradientu, co służy do dalszych obliczeń.

##### 2. Wykrycie miejsc występowania krawędzi.

W tym celu został wykorzystany algorytm non-max suppression. Polega on na wyborze

### 3.4. OPRACOWANY ALGORYTM PÓŁAUTOMATYCZNY

takich pikseli, które mają największą wartość gradientu na linii o kierunku zgodnym z kątem danego gradientu. Możliwe są 4 kierunki: pionowy, poziomy oraz dwa diagonalne. Jeśli dany piksel miał większą wartość gradientu od dwóch swoich sąsiadów, to zaznaczono go jako potencjalny punkt tworzący krawędzie. W ten sposób otrzymano obraz z potencjalnymi krawędziami.

#### 3. Wyznaczanie krawędzi progowaniem histerezy

Po poprzednim kroku na obrazie nadal znajdują się nieistotne krawędzie. W tym celu Canny wprowadził ideę progowania histerezy. Metoda ta wymaga 2 wartości progowych  $T_1, T_2$  takich, że  $T_1 < T_2$ . Jeżeli wartość gradientu w danym pikselu jest większa od  $T_2$ , to zaznaczono ten punkt jako krawędź. Jeśli tak się stało, to zaczęto proces śledzenia krawędzi — dla każdego sąsiada, którego wartość gradientu jest większa od  $T_1$  zaznaczono go jako krawędź. Jest ona wykonywana rekurencyjnie dla każdego zakwalifikowanego punktu.

Zamiast dokładnych wartości progowych można przekazać do funkcji 2 wartości —  $t_1, t_2$ , które są procentem liczby pikseli, które będą niedopuszczone jako krawędzie. Dla  $t_1 = 0.7, t_2 = 0.9$  dopuszczono tylko 10% pikseli jako te, które są większe od  $T_2$ . Podając  $t_1, t_2$  wyznaczono rozkład wartości gradientu w badanym obrazie, obliczono dystrybuantę  $F(x)$  i wybrano dla  $T_1$  ten argument, dla którego  $F(x) = t_1 * p$ , gdzie  $p$  to liczba pikseli i analogicznie dla  $T_2$ . W ten sposób wyznaczono progi do histerezy.

Operator Sobela [?] to metoda wyznaczania gradientu, a więc zarazem krawędzi zarówno w kierunku poziomym, jak i w pionowym. Dla każdego piksela przeprowadzono operację morfologiczną z następującymi maskami:

$$\begin{array}{ccc} \text{Maska rzędów} & & \text{Maska kolumn} \\ \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} & & \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \end{array}$$

Po wykonaniu tych operacji otrzymano wartości  $s_1$  i  $s_2$  odpowiednio dla maski rzędów i kolumn. Na podstawie tych danych otrzymano następujące informacje o gradiencie:

$$\begin{array}{cc} \text{Wielkość gradientu} & \text{Kierunek krawędzi} \\ \sqrt{s_1^2 + s_2^2} & \tan^{-1} \left[ \frac{s_1}{s_2} \right] \end{array}$$

Detektory krawędzi oparte na gradiencie, w tym operator Canny'ego są często używane. Ich główne zalety na podstawie [?] to:

### 3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

- dają dobre wyniki dla obrazów o dobrej jakości i bez szumów.
- są wydajne - ich złożoność jest liniowa względem liczby przetwarzanych pikseli.
- nie wymagają skomplikowanej sztucznej inteligencji do działania.

Zgodnie z [?] za ich główne wady można uznać:

- potrzebę określenia rozmiaru maski i wartości progowej, gdyż rozmiar maski znacząco wpływa na rozmieszczenie pozycji, w których gradient osiąga wartości maksymalne lub przecina zera,
- pomijanie narożników, które spowodowane jest faktem, że wartość 1D gradientu w narożnikach jest zazwyczaj zbyt mała, aby wykrywać wokół nich,
- znajdowanie schodkowych krawędzi, które są wykrywane tylko przez operator pierwszej pochodnej,
- dużą wrażliwość na szum,
- na podstawie obserwacji działania algorytmu — rozmyte krawędzie często nie są wykrywane przez małe różnice w wartościach kolejnych sąsiadujących pikseli.

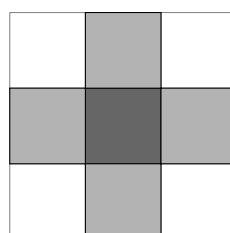
W ten sposób otrzymano macierz, gdzie każde pole w macierzy odpowiada pikselowi w wejściowej bitmapie — obrazie medycznym. Jeśli w komórce macierzy znajduje się 1, to w tym miejscu na bitmapie znajduje się krawędź, w przeciwnym przypadku 0. W ten sposób algorytm wykrył wszystkie znaczące krawędzie na bitmapie. Kolejnym krokiem przetwarzania było stworzenie grafu na podstawie wyżej wymienionej macierzy.

#### 3.4.2. Stworzenie grafu z bitmapy

Kolejnym etapem jest stworzenie grafu z bitmapy. Na tym etapie algorytm potrzebuje następujących danych wejściowych:

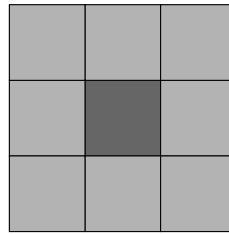
- macierz z wartościami logicznymi prawda/fałsz czy znajduje się w danym punkcie krawędź. Może to być także realizowane poprzez macierz wartości liczbowych.
- punkty wybrane przez użytkownika aplikacji.

Przed rozpoczęciem działania algorytmu należy zapewnić łączność 4-krotną (ang. Pixel 4-connectivity) [?]. Jest ona zwana także sąsiedztwem von Neumanna. Przy łączności 4-krotnej sprawdza się tylko sąsiadów w poziomie lub pionie.



### 3.4. OPRACOWANY ALGORYTM PÓŁAUTOMATYCZNY

Dla łączności 8-krotnej sprawdza się wszystkich możliwych sąsiadów, także po przekątnej. Jest ona zwana także sąsiedztwem Moore'a lub otoczeniem Moore'a [?].



W przypadku zastosowania łączności 8-krotnej przy wyznaczaniu długości krawędzi musiano by zastosować metrykę Czebyszewa, która jest specjalnym przypadkiem odległości Minkowskiego. Jeśli zostanie łączność 4-krotna to długość krawędzi byłaby obliczana zgodnie z metryką miejską, zwaną też metryką Manhattan.

Metryka Manhattan w kontekście dalszego przetwarzania w celu wyszukiwania najkrótszych ścieżek w grafie jest bardziej adekwatna, ponieważ jest intuicyjna w wyznaczaniu odległości na obrazie płaskim w porównaniu do metryki Czebyszewa. W tym przypadku najlepsza byłaby tutaj metryka Euklidesa, lecz mamy do czynienia nie z kolejnymi punktami oddalonymi od siebie, a z sąsiadującymi pikselami. Ponadto w tym algorytmie istotne jest szybkie szacowanie odległości, czy też długości danej krawędzi.

Wykrywanie wierzchołków przy łączności 4-krotnej jest prostsze. Wystarczy zliczyć liczbę sąsiadów. Poniżej zakładamy, że piksel jest oznaczony jako krawędź w macierzy wejściowej. W zależności od liczby sąsiadów mamy następujące przypadki:

- 0 — wierzchołek izolowany,
- 1 — punkty końcowe (ang. endpixels),
- 2 — punkty łączące (ang. linkpixels), czyli fragmenty krawędzi,
- 3–4 — punkty węzłowe (ang. vertices), czyli punkty, od których odchodzą co najmniej 3 krawędzie.

W przypadku łączności 8-krotnej do detekcji wierzchołków należałyby stosować przekształcenia Hit-or-Miss z elementami strukturalnymi. Elementy strukturalne do wykrywania odpowiednich punktów są następujące:

- wierzchołek izolowany:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

- punkty końcowe (ang. endpixels):

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ z & z & z \end{bmatrix},$$

- punkty łączące (ang. linkpixels), czyli fragmenty krawędzi, posiadają dokładnie 2 sąsiadów, nie używamy przekształcenia Hit-or-Miss a tylko liczymy sąsiadów

- punkty węzłowe (ang. vertices), czyli punkty, od których odchodzą co najmniej 3 krawędzie:

$$\begin{bmatrix} z & 1 & z \\ z & 1 & z \\ z & z & 1 \end{bmatrix} \text{ lub } \begin{bmatrix} 1 & z & z \\ z & 1 & z \\ 1 & z & 1 \end{bmatrix},$$

Warto zauważyć, że te elementy strukturalne należy obracać o 90, 180 i 270 stopni. Za każdym razem potrzeba wielokrotnie sprawdzać te same piksele. Ponadto należy sprawdzać 8, a nie 4 sąsiadów.

Kolejnym problemem jest fakt, że przy spójności 8-krotnej przekształcenie Hit-or-Miss może w najbliższym otoczeniu punktu krzyżowania się krawędzi oznaczyć kilka otaczających punktów, jako punkty węzłowe. Jest to złe rozwiązańe, ponieważ w ten sposób może nawet kilkukrotnie zwiększyć liczbę wierzchołków w grafie, co przełożyłoby się na niską wydajność algorytmu.

Ostatnim problemem z jakim należało się wiązać wybierając łączność 8-krotną jest fakt, że macierz wejściową dla tego etapu algorytmu należało poddać procesowi szkieletyzacji. Najlepiej byłoby w tym celu skorzystać z algorytmu KMM [?] lub K3M [?]. Te algorytmy musiałyby co najmniej raz przejrzeć całą macierz z wykrytymi krawędziami w optymistycznym przypadku.

Z wyżej wymienionych powodów zdecydowano się na łączność 4-krotną. Przygotowano i zaimplementowano algorytm tworzący graf z bitmapy, a jego pseudokod znajduje się w Załączniku 1.

Utworzony w ten sposób graf jest grafem nieskierowanym z wagami, gdzie wagi to liczba pikseli, czy też punktów należących do krawędzi. Graf ten może nie być spójny. Ponadto ten graf może nie zawierać wierzchołków, które pokrywają się z punktami wybranymi przez użytkownika.

Graf ten zazwyczaj jest rzadki, ponieważ liczba jego krawędzi jest rzędu liczby jego wierzchołków. Z uwagi na czasami bardzo dużą liczbę wierzchołków — nawet do kilkudziesięciu tysięcy — próba implementacji przy pomocy macierzy sąsiedztwa mogłaby spowodować zużycie całej możliwej pamięci operacyjnej. Dla 50 tysięcy wierzchołków program musiałby zadeklarować macierz sąsiedztwa zawierającą 2,5 miliarda komórek. Z tych powodów graf został zaimplementowany przy pomocy list sąsiedztwa.

### 3.4. OPRACOWANY ALGORYTM PÓŁAUTOMATYCZNY

W przyjętej implementacji każda krawędź zawiera dodatkowo informację o tym, jakie piksele należą do danej krawędzi w rzeczywistym obrazie.

#### 3.4.3. Zapewnienie spójności grafu

W pierwszym kroku na tym etapie przetwarzania grafu są dodawane punkty wybrane przez użytkownika do grafu.

Graf, który uzyskano w poprzednim kroku może nie być spójny. Powoduje to fakt, że między punktami wybranymi przez użytkownika mogą nie istnieć ścieżki. W celu zapewnienia spójności grafu należy dodawać sztuczne krawędzie. Zostają one dodawane z większymi wagami, niż wynikłoby to w rzeczywistości z liczebności listy pikseli, które reprezentują, ponieważ ma to na celu używanie z większym priorytetem prawdziwych krawędzi, a nie sztucznych. W sytuacji gdy nie istnieje dobra ścieżka z prawdziwych krawędzi, to zostaną użyte krawędzie sztuczne. Duże wagi dodatkowo będą zmuszały algorytmy wyszukiwania najkrótszych ścieżek w grafie do minimalizowania długości takich fragmentów zawierających sztuczne krawędzie.

Do znajdowania spójnych składowych grafu najczęściej używa się albo algorytmu przeszukiwania grafu w głąb (ang. Depth-first search, w skrócie DFS) lub przeszukiwania grafu wszerz (ang. Breadth-first search, w skrócie BFS) [?]. Użyto algorytmu przeszukiwania wszerz, opierając się na przykładowej implementacji w materiałach [?]. Wykorzystano Queue<T>, a zatem kolejkę, więc jest to przeszukiwanie wszerz. Złożoność obliczeniowa tego algorytmu to  $O(|E|)$ .

Po wyznaczeniu spójnych składowych grafu dla każdej pary składowych znajdowano taką parę wierzchołków, że odległość między nimi jest minimalna. Jeśli ta odległość była mniejsza niż maksymalna odległość między dwoma dowolnymi punktami zaznaczonymi przez użytkownika, to była dodawana sztuczna krawędź o wadze 2,5 razy większej niż odległość wynikająca z metryki Manhattan pomiędzy tymi dwoma wierzchołkami.

Przykład zasady działania tej operacji przedstawia poniższy pseudokod:

```
1) foreach (Spójna składowa grafu s1)
2) {
3)     foreach (Spójna składowa grafu s2, różna od s1 i nie przetwarzana
            wcześniej jako s1)
4)     {
5)         Wybierz wierzchołek v1 z s1 i v2 z s2 takie, że odległość pomiędzy
            nimi jest najmniejsza ze wszystkich takich par
6)         Dodaj sztuczną krawędź pomiędzy v1 i v2
7)     }
```

8) }

W ten sposób osiągnięto spójny graf, który zawiera punkty dodane przez użytkownika.

#### 3.4.4. Wyszukanie najkrótszych ścieżek w grafie

Na tym etapie została zapewniona spójność grafu. Z całą pewnością istnieje ścieżka pomiędzy punktami wybranymi przez użytkownika, te punkty są osiągalne. Pozostało wybrać najkrótszą ścieżkę łączącą kolejne te punkty. Założono, że jest dana lista punktów wybranych przez użytkownika i zawiera ona kolejne punkty, tzn. pierwszy należy połączyć z drugim, drugi z trzecim, . . . , ostatni z pierwszym.

Warto zaznaczyć, że wyszukiwano ścieżki o minimalnej wadze, czy też o minimalnym koszcie. Z uwagi na fakt, że wagi w grafie są ściśle związane z odległościami to używane jest sformułowanie szukania najkrótszych ścieżek. Wagi w tym grafie są nieujemne. Może istnieć w tym grafie kilka ścieżek z jednego punktu do drugiego o tym samym koszcie, więc algorytm kończy swoje działanie na tym etapie, gdy znajdzie jedną z nich. W tym grafie nie występują krawędzie wielokrotne. Ten graf nie zawiera cykli własnych.

Do wyszukiwania najkrótszych ścieżek w grafie po zgłębieniu literatury [?] i [?] były rozważane do użycia 2 algorytmy. Był to algorytm Dijkstry i A\*. Do algorytmu A\* była rozważana heurystyka w postaci odległości miejskiej, Manhattan z wierzchołka  $v$  do celu  $t$ , ozn.  $h(v)$ .

Zgodnie z [?] „Funkcja  $h$  musi spełniać następujące warunki:

- musi być oszacowaniem dolnym, czyli dla każdego wierzchołka  $v$   $h(v) \leq$  odległość  $v$  od celu  $t$ ,
- musi być monotoniczna, czyli dla dowolnej krawędzi  $\langle u, v \rangle$   $h(u) \leq$  waga  $\langle u, v \rangle + h(v)$ .

Heurystyka w postaci metryki Manhattan spełnia te wymagania. Z tego powodu może zostać wykorzystany algorytm A\*. Porównując złożoności algorytmu A\* i Dijkstry w [?] możemy zauważać, że algorytm A\* w pesymistycznym przypadku ma taką samą złożoność jak algorytm Dijkstry, czyli  $O(|E| * \log(|V|))$  z kolejką priorytetową dla grafów rzadkich, a  $O(|V|^2)$  nie wykorzystując kolejki priorytetowej dla grafów gęstych. W praktyce dla grafów rzadkich nie ma potrzeby rozważania znacznej części wierzchołków, co poprawia złożoność średnią. Wynosi ona wtedy  $O(|E|)$ .

Do implementacji wyszukiwania najkrótszych ścieżek w grafie na podstawie wyżej wymienionych wniosków został wykorzystany algorytm A\* z heurystyką w postaci metryki Manhattan. Przykładowy pseudokod tego algorytmu można znaleźć w [?] i jest on następujący:

### 3.4. OPRACOWANY ALGORYTM PÓŁAUTOMATYCZNY

```

1) CLOSE = 0 // zbiór (z szybkim sprawdzeniem przynależności)
2) OPEN = {s} // kolejka priorytetowa
3) // priorytety - sumy odległości wierzchołków od źródła
4) // i oszacowań odległości tych wierzchołków od celu
5) odległość[s] = 0
6) while ( OPEN niepusty )
7) {
8)     u = wierzchołek należący do OPEN taki, że
9)         odległość[u] + oszacowanie[u,t] <=
10)            odległość[w] + oszacowanie[w,t]
11)        dla wszystkich w należących do OPEN
12)        usuń u z OPEN
13)        wstaw u do CLOSE
14)        if ( u == t ) break
15)        foreach ( wierzchołek w sąsiadujący z u taki, że w należący do CLOSE )
16)        {
17)            if ( w należy do OPEN )
18)            {
19)                odległość[w] = nieskończoność
20)                wstaw w do OPEN
21)            }
22)            if ( odległość[w] > odległość[u] + waga<u,w> )
23)            {
24)                odległość[w] = odległość[u] + waga<u,w>
25)                aktualizacja priorytetu wierzchołka w (w OPEN)
26)                poprzedni[w] = u
27)            }
28)        }
29)    }

```

Aby maksymalnie dobrze wykorzystać niską złożoność obliczeniową algorytmu A\* należało wykorzystać dobre struktury danych do tego algorytmu. Zbiór CLOSE wymagał szybkiego sprawdzania przynależności. Po przenalizowaniu rekomendowanych struktur danych [?] został użyty HashSet<T>.

Zbiór Open oferował zastosowanie znacznie bardziej finezyjnych struktur danych. Wymagane

### 3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

było od niego, aby był kolejką priorytetową, czyli aby był sortowany po priorytetach będącymi sumą odległości wierzchołków od źródła i oszacowań odległości tych wierzchołków od celu. Warto zauważać, że oprócz sortowania często są wstawiane do niego wierzchołki, pobierane, usuwane i modyfikowane wartości priorytetu. Bardzo często jest sprawdzana przynależność i jest wyszukiwanie według klucza.

W [?] na stronie 317 znajduje się tabela 7.1. Przedstawia ona porównanie czasów różnych operacji dla różnych klas słownikowych zaimplementowanych w technologii .NET. Najlepszą strukturą danych byłoby SortedDictionary<K,V>, która jest implementowana przez drzewo czerwono-czarne gdyby nie fakt, że jest sortowane po kluczu, a nie tak jak jest potrzebne w tym przypadku sortowanie po wartości. Poszukiwano zatem struktury danych, która sortuje po wartościach i ma łatwy dostęp przez klucz.

W naszym rozwiążaniu została zaimplementowana struktura danych opierająca się na dwóch podstrukturach - zaimplementowanej kolejki priorytetowej poprzez listę, oraz słownik Dictionary<K,V> z technologii .NET, który opiera się o Tablicę skrótów. Zaimplementowana lista miała następującą strukturę:

```
public class MySortedListElement
{
    public Vertex Key;
    public double Value;
    public MySortedListElement next;
    public MySortedListElement previous;
}
```

Struktura ta miała zaimplementowane sortowanie przez wstawianie, zatem modyfikacja tylko jednego elementu wymagała w pesymistycznym przypadku tylko  $O(n)$  porównań. Wstawianie nowego elementu ma złożoność pesymistyczną  $O(n)$ , usuwanie  $O(1)$ . Zbadanie przynależności po kluczu to  $O(n)$  i nie zaleca się tego robić.

Niezależnie od tej struktury jest przechowywana druga struktura danych, Dictionary<K,V>, gdzie kluczami są wierzchołki, a wartości to referencje na elementy tej listy. Zmienne odpowiedzialne za przechowanie kolejki priorytetowej OPEN zostały zadeklarowane w sposób następujący:

```
Dictionary<Vertex, MySortedListElement> OpenDictionary =
    new Dictionary<Vertex, MySortedListElement>();
MySortedList OpenList = new MySortedList();
```

### 3.4. OPRACOWANY ALGORYTM PÓŁAUTOMATYCZNY

W ten sposób w połączeniu tych dwóch podstruktur otrzymujemy strukturę danych o następujących własnościach:

- dodawanie w czasie  $O(n)$ ,
- wyszukiwanie po kluczu w czasie  $O(1)$ ,
- wybierz najmniejszy element według wartości w czasie  $O(1)$ ,
- usuwanie w czasie  $O(1)$
- modyfikowanie jednego elementu w czasie  $O(n)$ .

Algorytm A\* jest uruchamiany dla każdego punktu wybranego przez użytkownika. Wyszukuje on najkrótszą ścieżkę do kolejnego punktu wybranego przez użytkownika. Po wszystkich obliczeniach obrys składa się z poszczególnych ścieżek. Są one konsolidowane i zwarcane jako lista pikseli na podstawie danych z krawędzi o listach pikseli, z których jest zbudowana krawędź. W ten sposób jest wykrywany obrys pomiędzy punktami zaznaczonymi przez użytkownika.

W sytuacji, gdy pomiędzy różnymi wierzchołkami nie istnieją prawdziwe krawędzie, albo istnieją tylko w wybranym fragmencie ścieżki łączącej te dwa wierzchołki, to algorytm w miarę możliwości będzie starał się używać prawdziwych krawędzi, dzięki odpowiedniej wadze krawędzi sztucznych. Dzięki sztucznym krawędziom na pewno istnieje droga między kolejnymi punktami, zatem algorytm z całą pewnością zwróci poprawny wynik. Co najwyżej będzie to wielokąt z punktami wybranymi przez użytkownika.

Algorytm dla sztucznych krawędzi generuje listę pikseli algorytmem Bresenhama [?]. Jest to algorytm, który dodaje w najbardziej optymalny sposób krawędzie. W celu zapewnienia łączności 4-krotnej jest przeprowadzana operacja morfologiczna z wykrywaniem 2 pikseli po przekątnej z maską 4-pikselową. Gdy zostaną takie piksele wykryte, to na jednym z rogów jest dodawany piksel w celu zapewnienia łączności 4-krotnej.

#### 3.4.5. Optymalizacja

Większość plików medycznych ma rozdzielczości rzędu kilkaset na kilkaset pikseli, a więc na całym obrazie znajduje się kilkaset tysięcy pikseli. Zdarzają się też pliki w znacznie większej rozdzielczości, a takimi jest między innymi mammografia i zdjęcie rentgenowskie. Na tych obrazach znajduje się do kilkudziesięciu milionów pikseli. W celu sprawnego przetwarzania tych informacji potrzebne były optymalizacje do algorytmu. Zostały dodane usprawnienia, które opisano poniżej.

##### 1. Zmniejszenie rozmiaru obrazu roboczego

W pierwszym kroku zmniejszono rozmiar obrazu roboczego. Wyznaczono najmniejszy prostokąt, w którym mieścią się wszystkie punkty zaznaczone przez użytkownika. Powiększono ten prostokąt o marginesy w taki sposób, by wyjściowy prostokąt nadal był w środku,

### 3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

a pole powiększonego było 4-krotnie większe. Innymi słowami — dodano marginesy po 50% szerokości / wysokości do każdego wymiaru. Jeśli rozmiar powiększonego prostokąta jest większe niż obrazu, to nie jest zmieniany obraz roboczy. Optymalizując w ten sposób w sytuacji, gdy użytkownik stworzył mały obrys, to rozmiar przetwarzanego obrazu od momentu operatora Sobela jest już najczęściej kilkukrotnie mniejszy, czasami nawet kilkadesiąt razy. W sytuacji, gdy obrys zajmuje prawie cały obraz medyczny, to nadal musi być przetwarzany cały obraz.

#### 2. Sięganie do pamięci zamiast to bitmapy

W celu przyśpieszenia działania operatora Sobela i liczenia statystyk zamiast sięgać do bitmapy metodą `.GetPixel()` na platformie .NET bitmapa została zapisana do tablicy bajtów i był z niej bezpośredni dostęp. Została wykorzystana do tego funkcja `.LockBits()`. Zysk na tej operacji był kilkudziesięciokrotny. Na późniejszych etapach algorytmu pracowano na macierzy, która nie była już bitmapą, więc operacje te wykonywały się znacznie szybciej.

#### 3. Dzielenie zbyt długich krawędzi

W sytuacji, gdy wczytane krawędzie do grafu są bardzo długie, to może zdarzyć się taka sytuacja, że punkt zaznaczony przez użytkownika znajduje się bardzo blisko krawędzi, ale daleko od punktu początkowego lub końcowego krawędzi. W tym celu jest wprowadzony mechanizm dzielenia zbyt długich krawędzi na kilka krótszych.

#### 4. Problem z ilością punktów

Dla każdego punktu jest uruchamiany algorytm A\*, zatem w sposób liniowy od ilości punktów zależy liczba uruchomień algorytmu A\*. Nie można wyznaczyć, czy w złożoności średniej całego algorytmu algorytm jest zależny liniowo od ilości punktów, czy w sposób logarytmiczny, czy też w sposób stały.

#### 3.5. Moduł obliczeń statystyk

System zapewnia liczenie statystyk dotyczących obrysu. Statystyki są identyczne zarówno dla obrysów manualnych i półautomatycznych, do których należą:

- histogram — wykres pokazujący natężenie kolorów w obrysowanym obszarze (kolor, to liczba w zakresie [0, 255], gdzie 0 oznacza najciemniejszy piksel a 255 najjaśniejszy),
  - wartość maksymalna — maksymalna wartość koloru piksela wewnątrz obrysu,
  - wartość minimalna — minimalna wartość koloru piksela wewnątrz obrysu,

### 3.5. MODUŁ OBLCZEŃ STATYSTYK

- wartość średnia — średnia arytmetyczna wartości pikseli wewnętrz obrysu,
- środek ciężkości — średnia arytmetyczna pozycji pikseli obrysu,
- długość obrysu w  $mm$  — obliczana na podstawie położenia pikseli i wartości pixel spacing odczytywanej z tagu obrazu DICOM,
- długość obrysu w pikselach,
- pole powierzchni w  $mm^2$ ,
- liczba pikseli wewnętrz obrysu.

Do obliczenia histogramu jest wymagany punkt wewnętrzny obrysu. Środek ciężkości obrysu nie zawsze musi znajdować się wewnętrz obrysu. W przypadku zastosowania algorytmu scan-linii dla obrysu manualnego nie jest dana informacja o logicznych wartościach, gdzie jest krawędź, więc nie można wykryć np. krawędzi poziomych. Algorytm scan-linii lepiej nadaje się w przypadku, gdy wypełniamy wielokąt, a nie obrys.

Z tego powodu wybrano algorytm Flood Fill, czyli rozlewania się rekurencyjnego zliczonych pikseli od punktu wewnętrznego. Właśnie z tego powodu od użytkownika jest wymagane podanie dodatkowej informacji, jaką jest punkt wewnętrzny obrysu. Założono, że użytkownik zaznaczy go poprawnie. W sytuacji, gdy zostanie podany błędny punkt to ten algorytm policzy to co znajduje się na zewnątrz obrysu, a nie to co jest wewnętrz.



## **4. Przeprowadzone eksperymenty**

Zostały przeprowadzno eksperymenty na stworzonym narzędziu do tworzenia obrysów. W podrozdziałach 4.1 - 4.4 są przedstawione wyniki tych eksperymentów.

Testy zostały przeprowadzone na komputerze stacjonarnym o następującej specyfikacji:

- procesor Intel Core i5-7500 o taktowaniu maksymalnym dla jednego rdzenia 3.8 Ghz,
- pamięć RAM Corsair Vengeance DDR4 16GB 3000 Mhz ustawiona w tryb 2140 Mhz o opóźnieniach CL15,
- płyta główna ASUS STRIX ROG Z270-I,
- procesor jest chłodzony powietrzem przez Noctue NH-L9i, procesor utrzymuje temperatury około 64 stopni Celsjusza przy temperaturze otoczenia około 21 stopni Celsjusza, nie występuje throttling,
- system operacyjny Windows 10.

### **4.1. Zbiór testowy**

Na zbiór testowy składały się najróżniejsze pliki. Po pierwsze - są to pliki, które posiada i do których ma prawo użycia Tomasz Świerczewski - jeden z autorów tego narzędzia do tworzenia obrysów. Są to badania medyczne jego osoby lub Pana Wojciecha Świerczewskiego, ojca Tomasza Świerczewskiego.

Do testów zostały wykorzystane następujące badania:

1. badanie rezonansem magnetycznym barku prawego Tomasza Świerczewskiego około 8 tygodni po zwichnięciu stawu ramiennego typu przedniego podkruczego,
2. zdjęcie rentgenowskie barku prawego Tomasza Świerczewskiego w momencie zwichnięcia stawu ramiennego,
3. badanie tomografią komputerową kręgosłupa Pana Wojciecha Świerczewskiego,
4. badanie rezonansem magnetycznym kręgosłupa Pana Wojciecha Świerczewskiego.

Kolejno te badania będą nazywane Badaniem 1. i tak dalej. Pliki te były wykonane na różnych

maszynach, w różnych szpitalach, różnie były zapisywane pliki DICOM. Są to różne badania, od rezonansu magnetycznego do RTG.

W ramach testów aplikacji oprócz testowania na tych plikach, zostały przeprowadzone testy również na plikach dostarczonych przez Panią Promotor. Zawierały one badania organów wewnętrznych, takich jak wątroba i trzustka jako badanie rezonansem magnetycznym. Dodatkowo znajdowała się tam mammografia.

## 4.2. Analiza działania aplikacji

Kolejne badania były otwierane przez stworzoną aplikację. Rysunek ?? przedstawia otwarty obraz medyczny Badania 1. z dowolną serią i dowolną klatką, instancją. Rysunek ?? jest odpowiednikiem Badania 2., Rysunek ?? Badania 3., Rysunek ?? Badania 4.

Ten test pozwolił sprawdzić, że aplikacja otwiera różne badania medyczne, od rezonansu magnetycznego, przez tomografię komputerową po zdjęcie rentgenowskie. Pokazano tutaj tylko kilka rodzajów badań, choć aplikacja obsługuje też inne np. mammografię. Na każdym z tych obrazów medycznych w wyniku przeprowadzonych testów można robić obrys. Przykładowy obrys ręczne i półautomatyczne znajdują się odpowiednio na Rysunku ?? i Rysunku ???. Ponadto można dostrzec, że są poprawnie generowane statystyki.

## 4.3. Wydajność algorytmu półautomatycznego

W celu sprawdzenia wydajności stworzonego algorytmu półautomatycznego posłużono się Badaniem 1., czyli badaniem rezonansu magnetycznego barku prawego.

W pierwszym teście postanowiono sprawdzić czasy generowania obrysu półautomatycznego kości ramiennej w zależności od ilości punktów. Oczekiwano, że obrys będzie wyglądał podobnie do tych z Rysunku ?? i ???. Był to obrys średniej wielkości. Rozmiar przetwarzanego obrazu w algorytmie prawdopodobnie się nie zmieniał i był to wycinek z obrazu medycznego zawierającego wszystkie te punkty i dodatkowe marginesy.

W trakcie dostawiania kolejnych punktów starano się poprawiać obrys w każdym kroku, tzn. aby wygenerowany obrys jak najlepiej odwzorowywał kość. Przy okazji dodawania kolejnych punktów do tego samego obrysu przetestowano funkcjonalność dodawania kolejnych punktów do obrysu.

Dla każdej liczby wierzchołków przeprowadzano 10 kolejnych pomiarów generowania tego

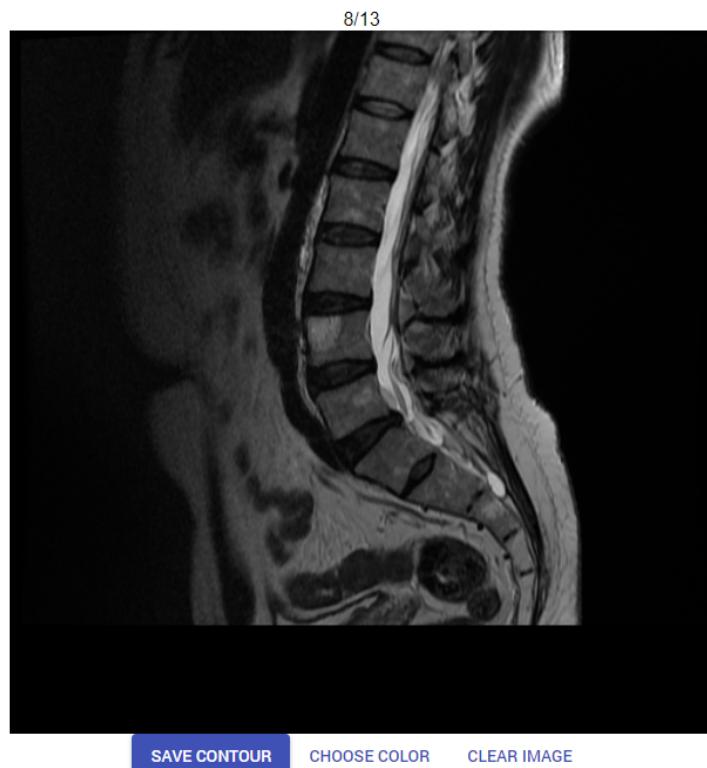
#### 4.3. WYDAJNOŚĆ ALGORYTMU PÓŁAUTOMATYCZNEGO



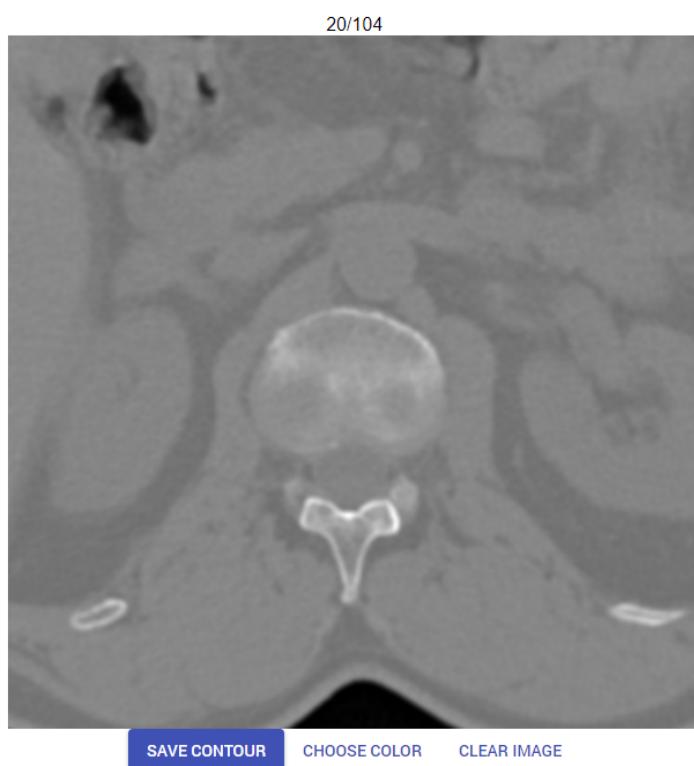
Rysunek 4.1: Widok badania rezonansem magnetycznym barku



Rysunek 4.2: Widok zdjęcia rentgenowskiego barku



Rysunek 4.3: Widok badania rezonansem magnetycznym kregosłupa



Rysunek 4.4: Widok badania tomografią komputerową kregosłupa

#### 4.3. WYDAJNOŚĆ ALGORYTMU PÓŁAUTOMATYCZNEGO

Testowy ręczny



**Perimeter:** 141.387534827084

**Area:** 1161.622819

**Center Of Mass:** [221,221]

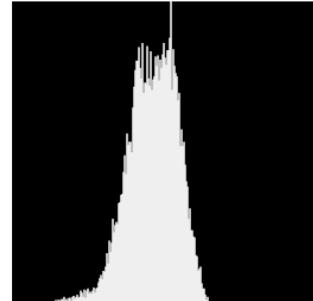
#### Histogram

**Histogram Max:** 169

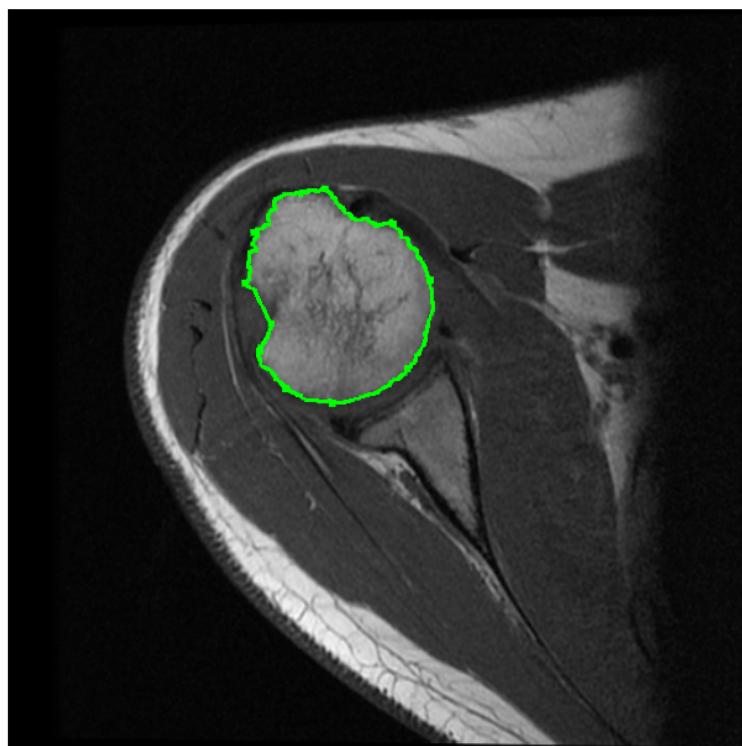
**Histogram Min:** 24

**Histogram Mean:** 118.307663882935

**Number Of Pixels Inside Contour:** 13531



Rysunek 4.5: Przykładowy obrys ręczny



**Perimeter:** 165.545

**Area:** 1169.778474

**Center Of Mass:** [223,223]

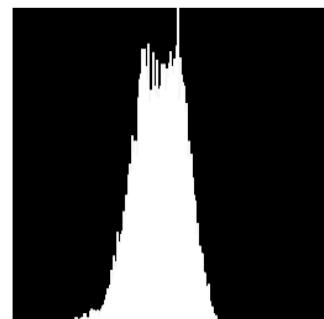
#### Histogram

**Histogram Max:** 169

**Histogram Min:** 36

**Histogram Mean:** 118.65419051812711

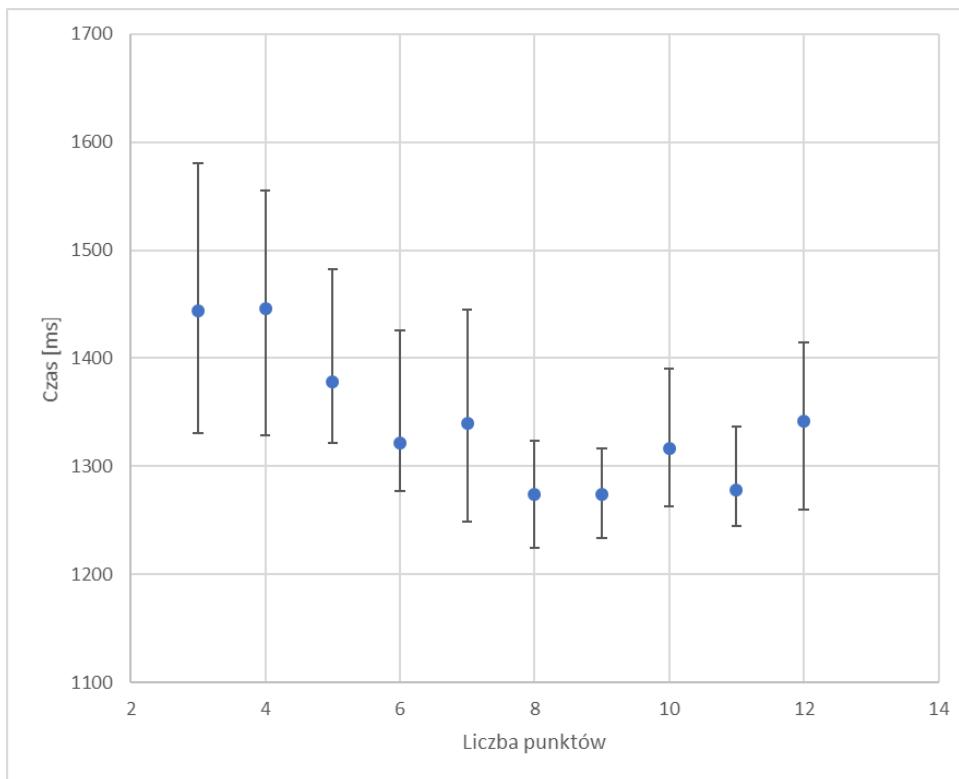
**Number Of Pixels Inside Contour:** 13626



Rysunek 4.6: Przykładowy obrys półautomatyczny

samego obrysu. Zostało to wykonane w celu niwelowania efektu różnego podporządkowania procesowi odpowiadającemu za obliczenia różnych priorytetów przez sam procesor. Ekstremalne wyniki zostały przedstawione w postaci słupków błędów, a średnia z 10 pomiarów jako niebieska kropka na wykresach.

Większość z generowanych niżej obrysów było podglądami, czyli bez użycia modułu statystyk. Gdy będzie inaczej, to zostanie to wyraźnie zaznaczone. Czas mierzono poprzez sprawdzanie czasu potrzebnego na wygenerowanie odpowiedzi przez API na otrzymane zapytanie.

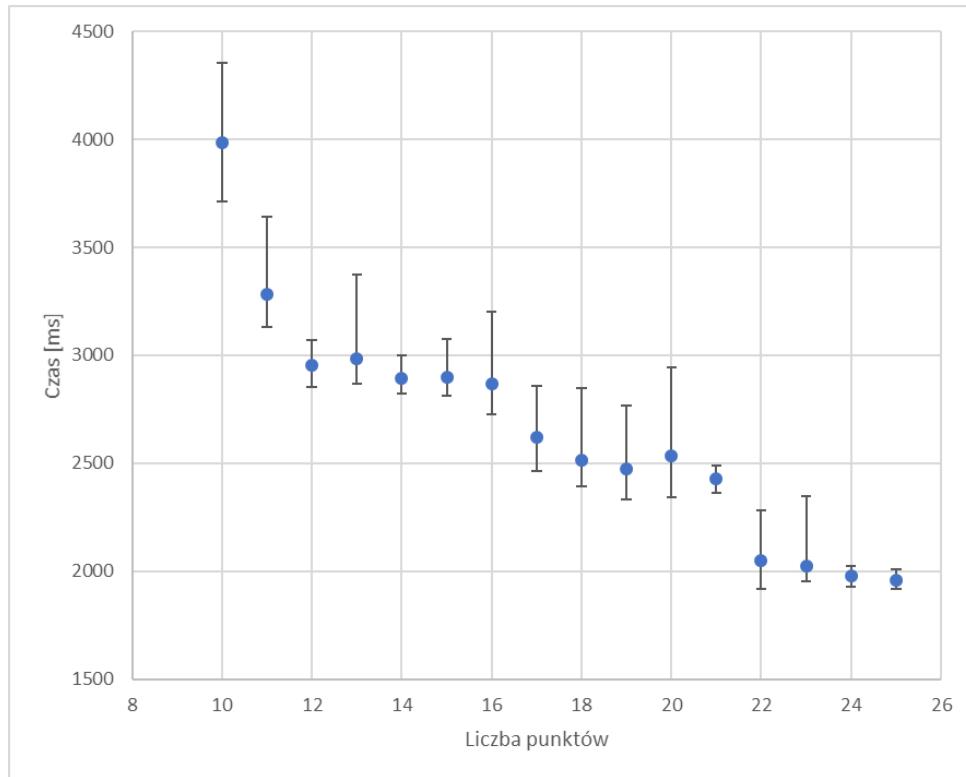


Rysunek 4.7: Czasy generowania obrysu półautomatycznego kości ramiennej w zależności od ilości punktów

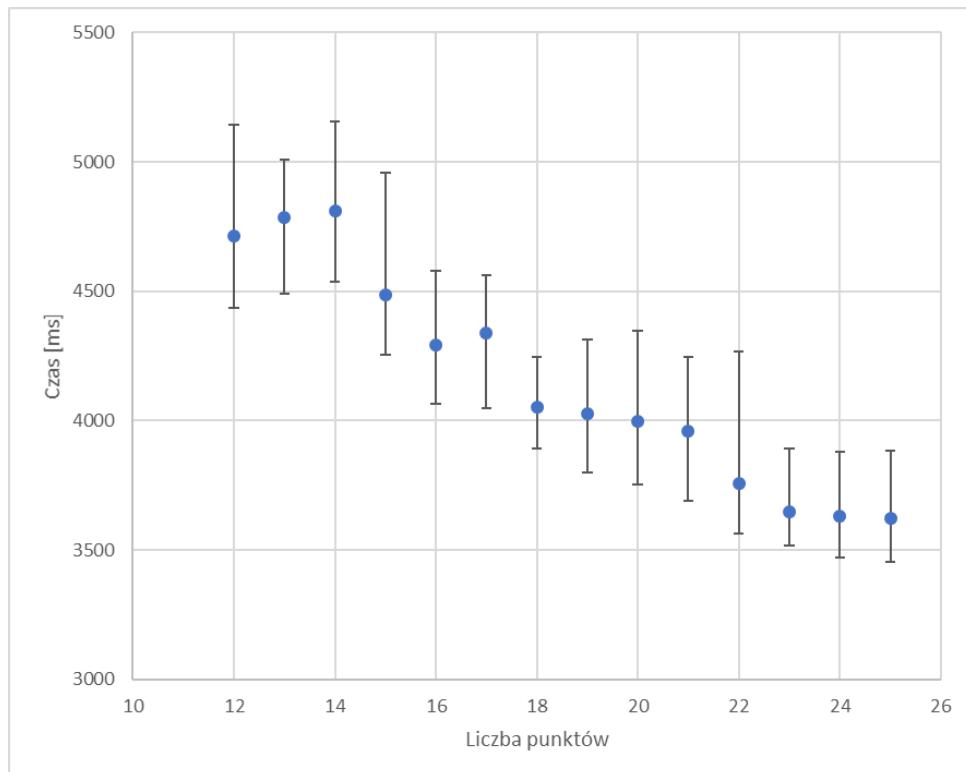
Na Rysunku ?? przedstawiono wyniki przeprowadzonych testów. Wraz z dostawianiem kolejnych punktów zmniejszał się czas potrzebny na wygenerowanie obrysu przez serwer. Warto zauważyć, że te czasy zmniejszyły się o około 150 ms. Nie dodawano kolejnych punktów z uwagi na osiągnięcie zadowalającej jakości wygenerowanego obrysu.

Na Rysunku ?? przedstawiono wyniki dla drugiego z testów. Starano się obrysować cały brak widoczny na obrazie medycznym. W tym celu stawiano kolejne punkty i obserwowało czasy jakie były potrzebne na wygenerowanie obrysu. Starano się wraz z kolejnymi dostawianymi punktami poprawiać generowany obrys. Czasami wybierano takie punkty, aby znalazły się pomiędzy kolejnymi dwoma najbardziej odległymi. Wygenerowany obrys zaprezentowano na Rysunku ???. Czasy zmieniły się prawie 2 krotnie, z poziomu około 4s do 2s.

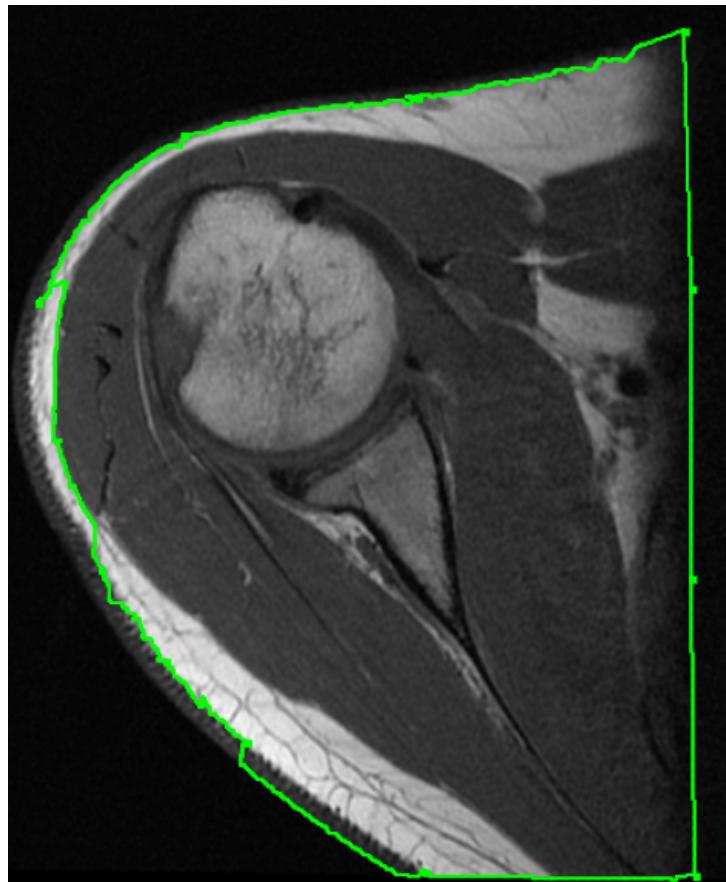
#### 4.3. WYDAJNOŚĆ ALGORYTMU PÓŁAUTOMATYCZNEGO



Rysunek 4.8: Czasy generowania obrysu półautomatycznego barku w zależności od ilości punktów



Rysunek 4.9: Czasy generowania obrysu półautomatycznego barku w zależności od ilości punktów drugą metodą



Rysunek 4.10: Jeden z wygenerowanych obrysów w trakcie testów

Na Rysunku ?? można zauważać czasami gwałtowne skoki w zmniejszającym się czasie potrzebnym na obliczenia. Mogło to być spowodowane faktem, że zmieniała się maksymalna odległość między kolejnymi punktami. Bliżej to zjawisko zostanie opisane w podrozdziale 4.4.

Na Rysunku ?? przedstawiono wyniki dla trzeciego testu. Starano się nie ingerować w wspomnianą wcześniej maksymalną odległość poprzez stawianie gęsto kolejnych punktów koło siebie na łuku. W ten sposób próbowało sprawdzić wpływ liczby uruchomień algorytmu A\* poprzez zmianę liczby punktów na czas obliczeń.

Uzyskane wyniki są bardzo podobne do poprzednich, choć bez wyraźnych skoków. Czas potrzebny na wygenerowanie obrysu nie spadł tak drastycznie jak w drugim teście, ale spadek był zauważalny, z około 4,7s do 3,7s.

Wraz z zwiększeniem się przestrzeni roboczej obrazu rośnie czas potrzebny na wygenerowanie obrysu. Na podstawie tych wykresów można wysnuć bardzo ciekawe wnioski, które zostaną zaprezentowane w podrozdziale 4.4.

#### 4.4. Analiza wyników i wnioski

Wyniki przeprowadzonych testów okazały się bardzo ciekawe. Po pierwsze, gdy generujemy statystyki, to czas potrzebny na obliczenie statystyk zwiększał czas na realizację zapytania o około 30%. Wiązało się to z odwiedzeniem wszystkich punktów wewnątrz obrysu.

Po drugie wraz z zwiększaniem rozdzielczości zdjęcia lub obrys stawał się coraz większy na obrazach medycznych o wysokiej rozdzielczości, to wzrastał proporcjonalnie czas potrzebny na wygenerowanie obrysu półautomatycznego. Wiązało się to z faktem, że został wykorzystany operator Sobela i Canny'ego, które wymagają odwiedzenia pewnej liczby pikseli, która odpowiednio się zwiększa. Dla mammografii średniej wielkości obrys był generowany nawet kilkadziesiąt sekund. Dla większości obrysów czasy obliczeń nie przekraczały zakładanych 5 sekund w wymaganiach niefunkcjonalnych.

Po trzecie wraz z zwiększaniem liczby punktów zmniejsza się czas potrzebny na wygenerowanie obrysu półautomatycznego. Powodowały to dwa oddzielne zjawiska. Pierwsze z nich, czyli maksymalna odległość pomiędzy kolejnymi dwoma punktami wybranymi przez użytkownika. W celu zapewnienia spójności grafu dodawano sztuczne krawędzie, o maksymalnej długości nieprzekraczającej maksymalnej odległości pomiędzy tymi wcześniej wspomnianymi dwoma punktami. Zatem im krótsza była to odległość, tym mniej było dodawanych sztucznych krawędzi. Liczba dodawanych sztucznych krawędzi spadała nawet kilkunastokrotnie. Dla algorytmu A\* oznaczało to kilkukrotnie mniejszą liczbę przetwarzanych krawędzi.

Drugim zjawiskiem odpowiedzialnym za zmniejszanie się czasu potrzebnego na wygenerowanie obrysu półautomatycznego wraz z zwiększającą się liczbą punktów był algorytm A\*. Początkowo przypuszczano, że zwiększanie liczby uruchomień algorytmu A\* wraz ze wzrostem liczby punktów będzie generował dodatkowy koszt obliczeniowy, a nie go redukował. Tak może się stać w pesymistycznym przypadku. W realnych obrysach wraz z dodawaniem kolejnych punktów w celu poprawienia jakości obrysu malała złożoność średnia, ponieważ algorytm A\* szybciej znajdował prawidłową ścieżkę, nie oddalał się bardzo daleko od wierzchołka źródłowego. Algorytm A\* dzięki heurystyce mógł pomijać bardzo dużą liczbę krawędzi, które na pewno nie polepszyłyby rozwiązania.

W ten sposób te dwa zjawiska powodowały, że wraz ze wzrostem liczby punktów wybranych przez użytkownika najczęściej malał czas potrzebny na wygenerowanie obrysu półautomatycznego. Czas dla większych obrysów maleje w sposób znaczący, dla małych obrysów tylko minimalnie. Dzieje się tak dlatego, że operacje na grafach są tylko elementem obliczeń, a wtedy dominującą częścią obliczeń są operacje na przetwarzanym obrazie. Ponadto te czasy zawierają

pewne nakłady czasowe, które były potrzebne na przetwarzanie zapytań.

Wnioski dla użytkownika powinny być takie, że użytkownik powinien unikać bardzo dużych obrysów na zdjęciach o bardzo dużej rozdzielczości, czyli np. mammografii czy zdjęcia rentgenowskiego. Ponadto użytkownik powinien oczekiwąć poprawnego działania dla zdjęć o dobrej jakości, ostrych krawędziach i o niskim szumie. W celu poprawienia zarówno jakości jak i wydajności użytkownik powinien stawiać więcej punktów niż mniej. Oczywiście w granicach rozsądku, czyli do kilkunastu zamiast wybierania tylko kilku punktów.

## 5. Podsumowanie

W podsumowaniu omówiono 3 aspekty: osiągnięcie celu pracy, napotkane problemy i ograniczenia.

Cel pracy został osiągnięty. Spełniono wszystkie wymagania funkcjonalne i niefunkcjonalne, z wyłączeniem wymagania dotyczącego wydajności generowania obrysu półautomatycznego dla obrazów DICOM o dużej rozdzielczości (ponad 1000000 pikseli).

W trakcie pracy nad systemem napotkano różne problemy i ograniczenia. Zostały one opisane poniżej.

W interfejsie ograniczeniem okazał się dostępny rozmiar ekranu. Projektowanie aplikacji zakładało wykorzystanie ekranu o rozdzielczości 1920x1080, a więc ekranu panoramicznego. Niestety obrazy DICOM mają bardzo zróżnicowane układy — są obrazy kwadratowe, pionowe oraz poziome. Założenie, że ekran użytkownika jest ekranem 1920x1080 sprawia, że obrazy pionowe będą wyświetlane jedynie w niewielkiej części obszaru przeznaczonego dla obrazów. Jest to niestety ograniczenie, którego nie da się zlikwidować, gdyż przy założeniu rozdzielczości 1080x1920 powoduje analogiczny problem z obrazami poziomymi. Zaleca się skonfigurowanie aplikacji pod wymagania konkretnej grupy lekarzy lub szpitala na podstawie wymiarów zdjęć, które będą częściej występować w systemie.

Podstawowym problemem występującym w trakcie implementacji rozwiązania było ustalanie kontraktów w warstwie komunikacyjnej. Ze względu na niewielką ilość akcji w komunikacji nie zdecydowano się na zastosowanie generatora kontraktów, ale zaleca się wprowadzenie takiego rozwiązania ponownie w trakcie rozwoju systemu. W zależności od złożoności komunikacji generator kontraktów może zdecydowanie usprawnić wprowadzanie zmian w aplikacji.

Podczas projektowania systemu zdecydowano, że obrazy DICOM będą wyświetlane w największej rozdzielczości umożliwiającej wyświetlenie całego obrazu na ekranie. W związku z tym, większość obrazów wyświetlona jest w rozmiarze różnym od faktycznego rozmiaru obrazu. Rozważano dwie możliwości rozdzielczości wykonywanych obrysów: faktyczne wymiary obrazu oraz rozmiary obrazu wyświetlonego na ekranie. Zdecydowano, że obrys powinny być zapisywane w wymiarach identycznych faktycznym rozmiarom obrazu. Decyzja została uzasadniona potrzebą zapewnienia poprawnego obliczania statystyk. Zapisywanie obrysów w takich wymiarach ułatwia

obliczanie liczby pikseli wewnątrz obrysu.

Ze względu na złożoność algorytmu używanego do wyznaczania obrysu półautomatycznego wykonywanie obrysów półautomatycznych w obrazach o dużej rozdzielczości przekracza dwukrotnie dopuszczalny czas ustalony w punkcie 5 wymagań niefunkcjonalnych. Czas obliczeń można poprawić poprzez ograniczenie obszaru, w którym wykrywane będą krawędzie, ale nie rozwiąże to problemu z wykonywaniem obrysów o wymiarach zbliżonych do pełnych wymiarów obrazu.

Podczas pracy nad systemem rozważano dodatkowe funkcjonalności, które nie zostały poruszone w tej pracy. Są one potencjalnymi możliwościami rozwoju stworzonego w ramach tej pracy systemu. Funkcjonalności opisano szczegółowo poniżej.

W obecnej wersji systemu nie zaimplementowano deskryptorów kształtu obrysu. Zimplementowanie takich deskryptorów mogłoby dostarczyć dodatkowe informacje na temat wykonanych przez użytkownika obrysów. Taka informacja może w przyszłości dostarczyć dodatkową zmienną, którą można by wykorzystywać w celu trenowania sztucznej inteligencji w zautomatyzowanym wykrywaniu organów, jak również wykrywaniu i sugerowaniu niepokojących zmian.

W tej wersji w systemie użytkownicy nie są rozróżnialni. Dodanie użytkowników pozwoliłoby na segregowanie tworzonych obrysów i wyświetlanie użytkownikowi obrysów wykonanych jedynie przez niego samego, a nie wszystkich obrysów istniejących w systemie. Z punktu widzenia użyteczności systemu użytkownik nie musiałby szukać swojego obrysu pośród obrysów innych użytkowników systemu.

Uwierzytelnianie zapobiegłoby również niepowołanemu dostępowi osób postronnych do wykonanych obrysów oraz uniemożliiłoby ataki typu DoS. W obecnej wersji można poprzez wysyłanie dużej liczby zapytań związanych z generowaniem obrysów półautomatycznych doprowadzić do niedostępności przeprowadzania akcji na serwerze. System jest również podatny na złośliwe działanie mające na celu zapełnienie całej dostępnej serwerowi przestrzeni dyskowej, które może zostać wywołane przez wysłanie dużej liczby zapisów obrysów manualnych.

Z punktu widzenia użytkownika interesujące mogą być informacje wyliczone przez system na temat wykonanych przez niego obrysów. W obecnej wersji systemu, jeśli użytkownik chciałby takie informacje zapisać musiałby samodzielnie przepisać dane wyświetlane w widoku szczegółów obrysu. Zautomatyzowanie takiej funkcjonalności mogłoby zaoszczędzić użytkownikowi wiele czasu.

Głównym celem wykonywania obrysów na obrazach medycznym jest generowanie zbioru testowego dla różnych metod sztucznej inteligencji mających na celu sugerowanie lekarzom niepokojących zmian wykrytych automatycznie. W związku z tym użytecznym rozszerzeniem funkcjonalności systemu byłoby zintegrowanie go z modułem sztucznej inteligencji i umożliwienie

obrysowywania wgrywanych obrazów metodami sztucznej inteligencji opartych na obrysach wykonanych przez użytkownika.



## Bibliografia

- [1] Albahari J., Albahari B.: C 6.0 w pigułce, *Helion, O'Reilly Media, Inc.*, Gliwice, page–page, 2016
- [2] Bresenham J. E.: Algorithm for computer control of a digital plotter. *ICM System Journal* 4(1) 1965
- [3] Bródka J.: Wykłady z przedmiotu Algorytmy i Struktury Danych 2 *Politechnika Warszawska, Wydział Matematyki i Nauk Informacyjnych* Materiały dostepne na stronie: <http://mini.pw.edu.pl/brodka/ASD2.html> [Dostęp 22 stycznia 2019]
- [4] Canny J. F.: Finding Edges and Lines in Images. *Technical report no. 720, Massachusetts Institute of Technology (MIT)*, Cambridge, Massachusetts, USA, 1983
- [5] Cytowski J., Gielecki J., Gola A.: Cyfrowe przetwarzanie obrazów medycznych: Algorytmy. Technologie. Zastosowania. *Akademicka Oficyna Wydawnicza EXIT*, Warszawa, 88–94, 2008
- [6] Facebook Inc.: Informacje o bibliotece ReactJS. Oficjalna strona: <https://reactjs.org/> [Dostęp 27 stycznia 2019]
- [7] Hafey C.: Dokumentacja projektu Cornerstone Core. Oficjalna strona: <https://docs.cornerstonejs.org/> [Dostęp 27 stycznia 2019]
- [8] Hart P. E., Nilsson N. J., Raphael B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths *IEEE Transactions on Systems Science and Cybernetics* 4(2), 100–107, 1968
- [9] ivmartel: Projekt DICOM Web Viewer. Oficjalna strona: <https://ivmartel.github.io/dwv/> [Dostęp 27 stycznia 2019]
- [10] Kronis K., Uhanova M.: Performance Comparision of Java EE and ASP.NET Core Technologies for Web API Developmnet. *Applied Computer Systems* 23, Riga Technical University, Ryga, 37–44, 2018

- [11] Microsoft Corporation: Dokumentacja platformy .NET. Oficjalna strona: <https://docs.microsoft.com/pl-pl/dotnet/> [Dostęp 22 stycznia 2019]
- [12] Microsoft Corporation: Dokumentacja rekomendowanych struktur danych platformy .NET. Oficjalna strona: <https://docs.microsoft.com/pl-pl/dotnet/standard/collections/> [Dostęp 22 stycznia 2019]
- [13] Microsoft Corporation: Informacje o platformie .NET Core. Oficjalna strona: <https://docs.microsoft.com/pl-pl/dotnet/core/about> [Dostęp 22 stycznia 2019]
- [14] Microsoft Corporation: Informacje o platformie ASP.NET Core. Oficjalna strona: <https://docs.microsoft.com/pl-pl/aspnet/core/?view=aspnetcore-2.2> [Dostęp 22 stycznia 2019]
- [15] Mozilla and individual contributors: Dokumentacja HTMLCanvasElement. Oficjalna strona: [https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/](https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement) [Dostęp 27 stycznia 2019]
- [16] National Electrical Manufacturers Association: Standard DICOM. Oficjalna strona: <https://www.dicomstandard.org/> [Dostęp 22 stycznia 2019]
- [17] Nowacki R., Plechawska-Wójcik M.: Analiza porównawcza narzędzi do budowania aplikacji Single Page Application — AngularJS, ReactJS, Ember.js, *Journal of Computer Sciences Institute 2, Politechnika Lubelska, Instytut Informatyki*, Lublin, 98–103, 2016
- [18] Osimis S.A.: Projekt Orthanc. Oficjalna strona: <https://www.orthanc-server.com/> [Dostęp 27 stycznia 2019]
- [19] Rosenfeld A., Kak A. C.: Digital Picture Processing, *Academic Press, Inc.*, Nowy Jork, 1982
- [20] Saeed K., Rybnik M., Tabędzki M., Adamski M.: Algorytm do Ścieniania Obrazów: Implementacja i Zastosowania *Zeszyty Naukowe Politechniki Białostockiej 2002 Informatyka - Zeszyt 1*, Białystok, 2002
- [21] Saeed K., Tabędzki M., Rybnik M., Adamski M.: K3M: A Universal Algorithm for Image Skeletonization and a Review of Thinning Techniques *International Journal of Applied Mathematics and Computer Science, 2010, 20(2)* Białystok, 317–335, 2010
- [22] Sedgewick R., Wayne K.: Algorytmy Wydanie IV, *Helion*, Gliwice, 526–706, 2012
- [23] Sobel I., Feldman G.: An 3x3 Isotropic Image Gradient Operator for Image Processing. *Presentation at Stanford Artificial Intelligence Project (SAIL) in 1968*, 2014

## BIBLIOGRAFIA

- [24] Weisstein, Eric W.: Moore Neighborhood. *From MathWorld—A Wolfram Web Resource.*  
<http://mathworld.wolfram.com/MooreNeighborhood.html> [Dostęp 22 stycznia 2019]



## Instrukcja instalacji

W celu instalacji serwera Orthanc należy otworzyć stronę <https://www.orthanc-server.com/download.php>, pobrać wersję odpowiednią dla używanego systemu operacyjnego, a następnie postępować zgodnie z instrukcjami wyświetlonymi podczas instalacji. W celu uzyskania szczegółowych informacji odnośnie konfiguracji należy zapoznać się z dokumentacją znajdującej się na stronie <http://book.orthanc-server.com/users/cookbook.html>

W celu instalacji serwera obrysów i aplikacji webowej należy skopiować pliki znajdujące się na płycie w folderach Web oraz DotNetProject na stację, na której aplikacje te zostaną uruchomione.

Do działania serwera obrysów wymagany jest .NET Core w wersji 2.2 lub nowszej. W celu instalacji .NET Core należy otworzyć stronę <https://dotnet.microsoft.com/download>, pobrać wersję odpowiednią dla używanego systemu operacyjnego, a następnie postępować zgodnie z informacjami wyświetlonymi w trakcie instalacji.

W celu konfiguracji serwera w pliku DotNetProject/Api/Properties/launchSettings.json należy wpisać w 24 linii wartość adresu i portu na którym serwer ma zostać udostępniony, jako wartość pola applicationUrl . Przykładowa poprawna wartość linii 24 to `applicationUrl": "https://localhost:5001",`.

Ponadto w pliku DotNetProject/Api/Startup.cs należy podać w linii 33 link pod którym dostępna jest aplikacja przeglądarkowa jako argument metody WithOrigins. Przykładowa poprawna wartość linii 24 to `builder => builder.WithOrigins("http://localhost:8080", "http://localhost:3000")`.

W celu uruchomienia serwera obrysów należy przejść do skopowanego folderu DotNetProject/API i wykonać w konsoli polecenie `$ dotnet build && dotnet run`.

Do działania aplikacji przeglądarkowej wymagany jest Node w wersji co najmniej 9.4 oraz yarn w wersji co najmniej 1.10.1, aby zainstalować Node.js należy udać się na stronę <https://nodejs.org/en/download/> i wybrać wersję odpowiadającą systemowi, z którego korzystamy. Postępować zgodnie ze wskazówkami instalatora. W celu instalacji yarn należy udać się na stronę <https://yarnpkg.com/en/docs/install> aby pobrać wersję odpowiadającą systemowi, na którym będzie uruchamiana aplikacja i uruchomić instalator.

W celu konfiguracji portu, na który będzie wystawiona aplikacja webowa należy ustawić od-

powiednią liczbę w 4 linii pliku Web/express.js. Przykładowa poprawna konfiguracja:

```
$ const portNumber = 3000;
```

Ponadto w celu konfiguracji aplikacji należy ustawić adresy do API serwera obrysów i serwera Orthanc w pliku Web/src/helpers/requestHelper.ts poprzez zmianę zawartości cudzysłowów w pierwszych dwóch linijkach pliku. Przykładowa poprawna konfiguracja:

```
export const orthancURL = "http://localhost:8042/";  
export const apiURL = "https://localhost:5001/";
```

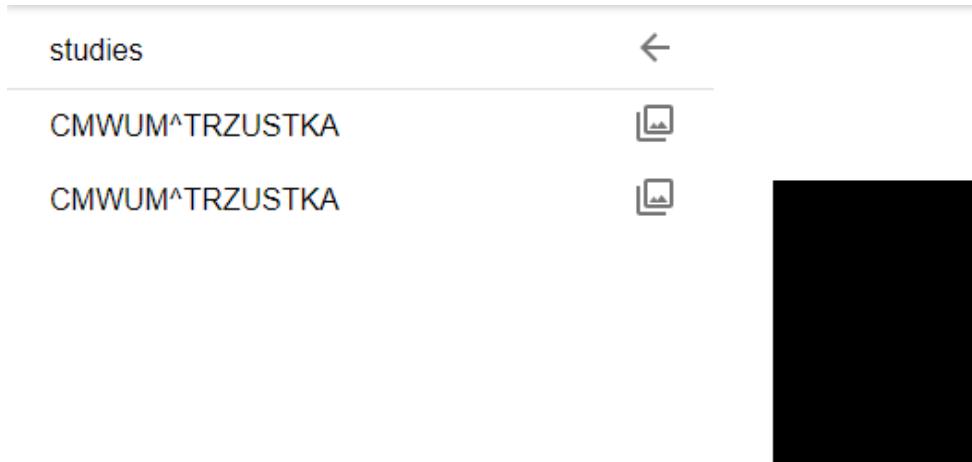
W celu uruchomienia aplikacji przeglądarkowej wchodzimy do folderu Web w którym wykonujemy polecenie `$ yarn install && yarn prod`. Uwaga, tę komendę należy uruchomić w konsoli obsługującej skrypty w języku bash.

## Instrukcja użytkowania

| DICOM contour              |   |   |
|----------------------------|---|---|
| patients                   |   |   |
| Anonymized1                | » | » |
| N/A                        | » | » |
| Mass-Test_P_00016_LEFT_CC  | » | » |
| Mass-Test_P_00016_LEFT_MLO | » | » |
| Mass-Test_P_00017_LEFT_CC  | » | » |
| Mass-Test_P_00017_LEFT_MLO | » | » |
| Sample patient name        | » | » |
| Sample patient name        | » | » |
| name                       | » | » |
| Tomka                      | » | » |

Rysunek 5.1: Widok listy pacjentów

Po otwarciu aplikacji użytkownik może przeglądać pacjentów (patients), których badania zostały wgrane do serwera Orthanc — Rysunek ??.



Rysunek 5.2: Widok listy badań pacjenta

Po wybraniu pacjenta poprzez kliknięcie lewym przyciskiem myszy na jego nazwę, na liście pojawiają się badania (studies) wybranego pacjenta. Użytkownik może wrócić do widoku pacjentów klikając w strzałkę na liście z badaniami— Rysunek ??.

## INSTRUKCJA UŻYTKOWANIA

| series                                  | < |
|---|---|
| t2_localizer_multi                      |   |
| t2_blade_cor_mbh_fs                     |   |
| PosDisp: [2] t2_blade_cor_mbh_fs        |   |
| t2_blade_tra_trigg_fs                   |   |
| PosDisp: [3] t2_blade_tra_trigg_fs      |   |
| t2+t2_tse_tra_mbh_p2                    |   |
| ep2d_diff_tra_b0_50_100_150_200_400_800 |   |
| ep2d_diff_tra_b0_50_100_150_200_400_800 |   |
| t1_fl2d_in_opp_ph_tra_mbh               |   |
| PosDisp: [9] t1_fl2d_in_opp_ph_tra_mbh  |   |
| t2_haste_cor_thin_slab_mbh_3mm          |   |

Rysunek 5.3: Widok listy serii w badaniu

Po wybraniu badania poprzez kliknięcie lewym przyciskiem myszy na jego nazwę, na liście pojawiają się serie (series) wybranego badania. Użytkownik może wrócić do widoku badań, klikając w strzałkę na liście z seriami — Rysunek ??.

---

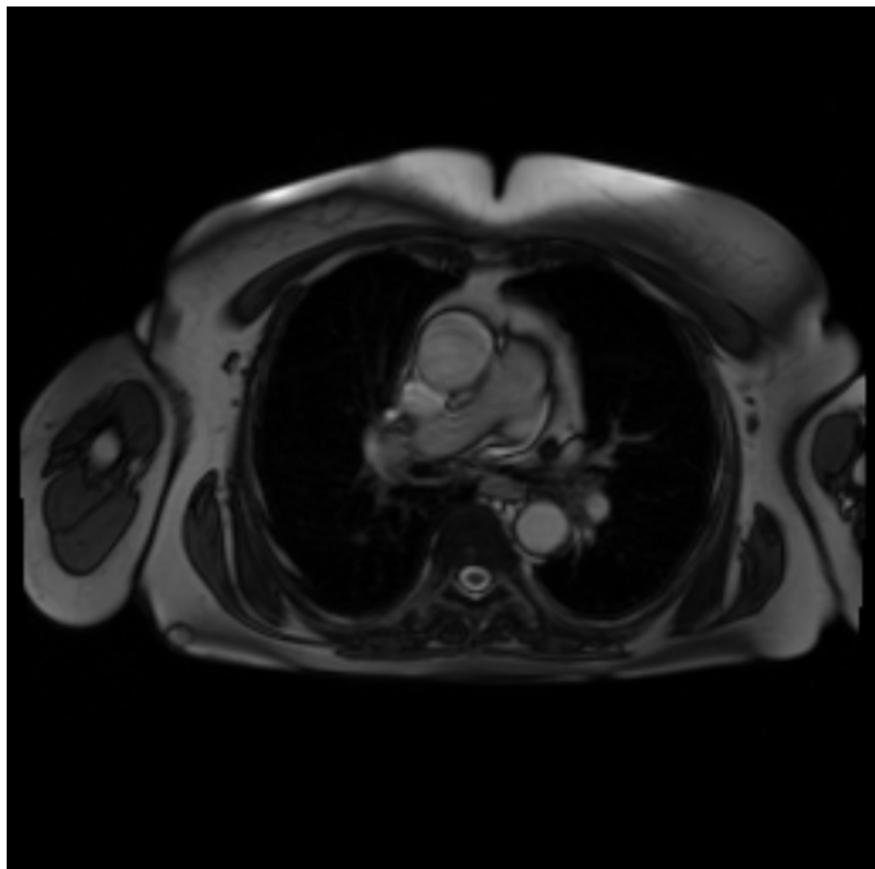
MANUAL

SEMI-AUTOMATIC

---

N/A/CMWUM^TRZUSTKA/t2\_localizer\_multi

7/15



SAVE CONTOUR

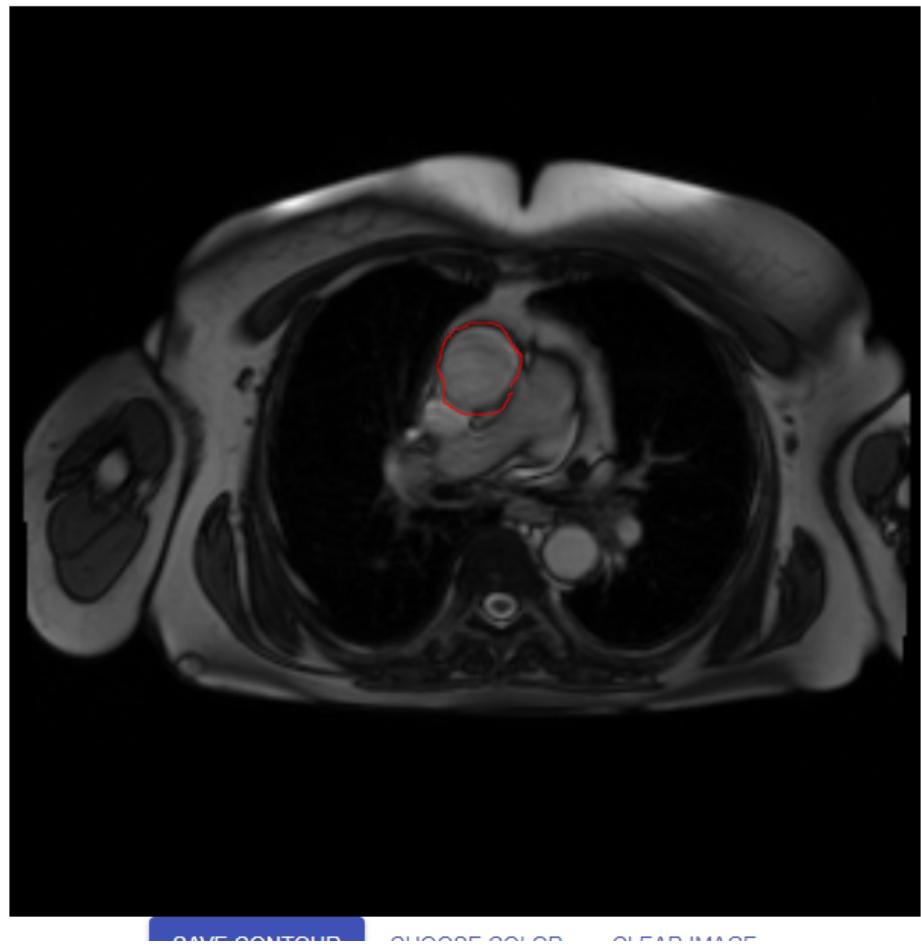
CHOOSE COLOR

CLEAR IMAGE

Rysunek 5.4: Wybrany obraz w module obrysów manualnego

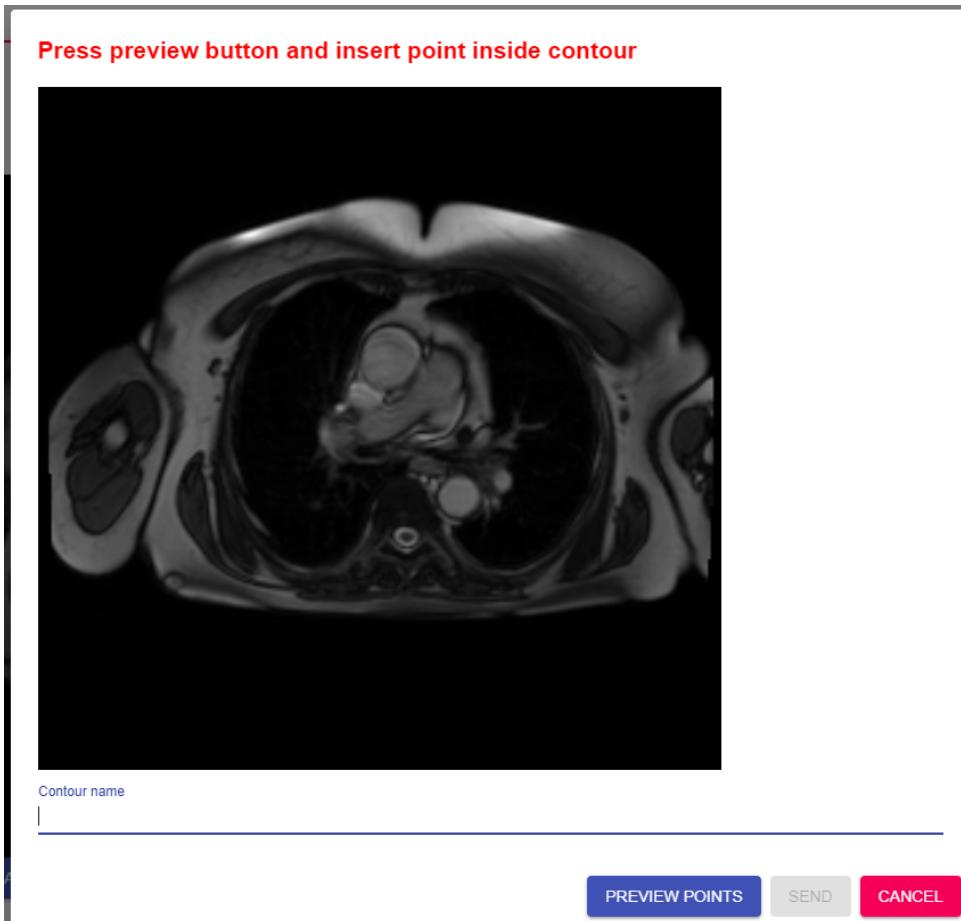
Po wyborze serii na ekranie pojawia się pierwszy obraz z serii. Użytkownik może przełączać się pomiędzy obrazami korzystając z listy obrazów po lewej stronie lub używając rolki myszy po najechaniu na obraz. Aby powrócić do wyboru serii należy kliknąć lewym przyciskiem myszy na strzałkę na liście instancji (instances) — Rysunek ??.

## INSTRUKCJA UŻYTKOWANIA



Rysunek 5.5: Przykładowy obrys

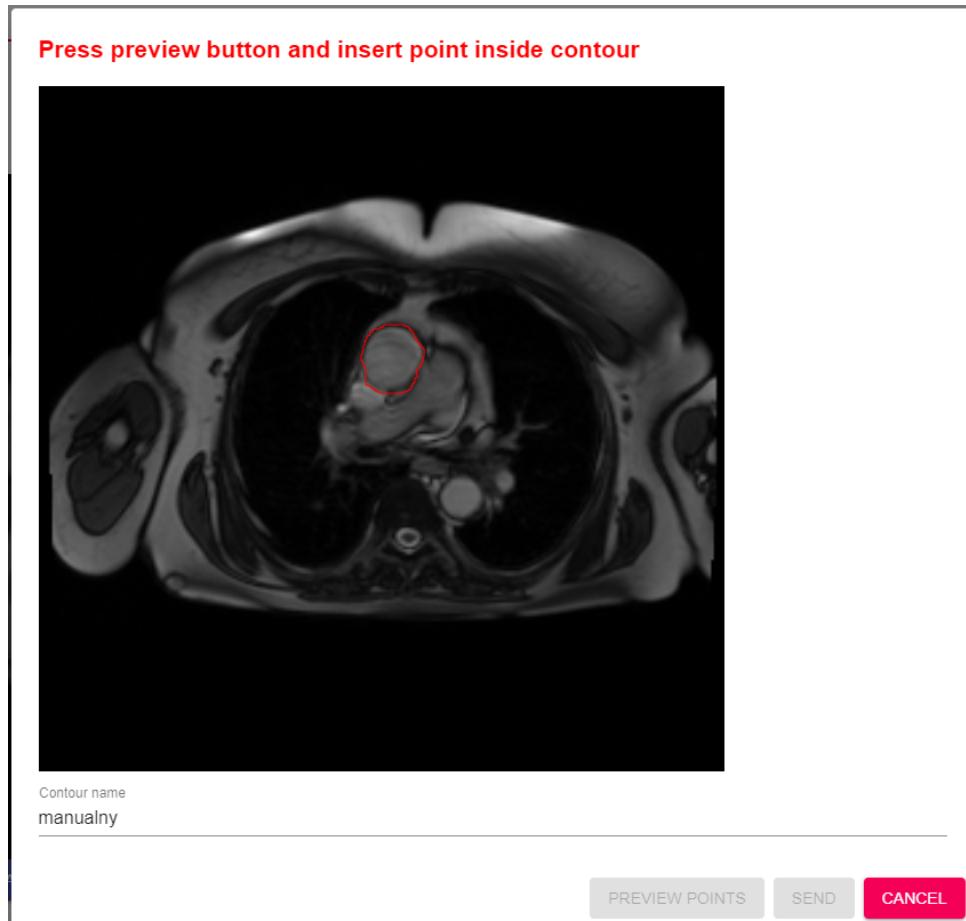
Przy manualnym obrysie użytkownik może wykonywać obrys poprzez przytrzymanie lewego przycisku myszy na obrazie, przesuwając mysz z wciśniętym przyciskiem. Gdy użytkownik korzysta z tabletu graficznego wystarczy, że będzie przesuwał wciśniętym rysikiem po tablecie — Rysunek ??.



Rysunek 5.6: Okno zapisu obrysu manualnego

Po wciśnięciu przycisku zapisz obrys (Save Contour), użytkownikowi ukazuje się okno dialogowe. W celu usunięcia nieudanego obrysu należy kliknąć lewym przyciskiem myszy w przycisk wyczyść obraz (Clear Image). W celu zapisania obrysu należy wykonać następujące kroki: Wpisać nazwę obrysu w pole tekstowe poniżej obrazu (pod napisem Contour Name) — ??.

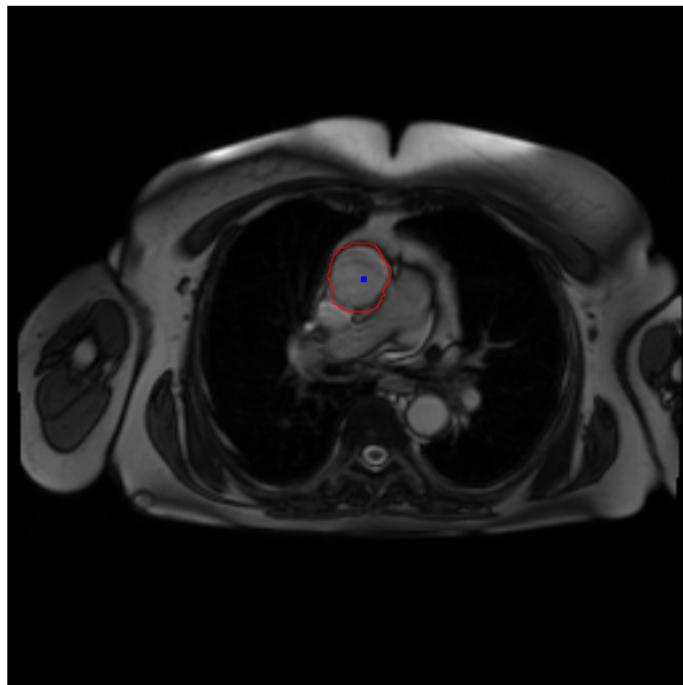
## INSTRUKCJA UŻYTKOWANIA



Rysunek 5.7: Podgląd wykonanego obrysu

Po wpisaniu nazwy, należy kliknąć przycisk podgląd punktów (Preview Points) lewym przyciskiem myszy. Pozwoli to użytkownikowi zobaczyć wykonany przez niego obrys — Rysunek ??.

**Press preview button and insert point inside contour**

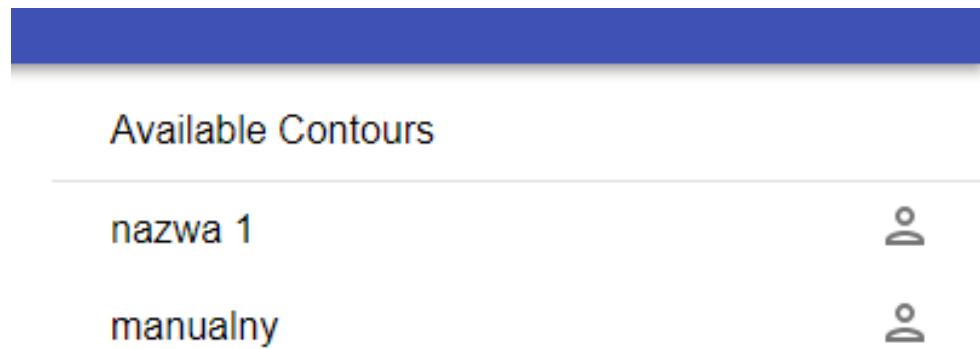


Contour name  
manualny

PREVIEW POINTS SEND CANCEL

Rysunek 5.8: Wybór punktu wewnętrz obrysu

Następną czynnością, którą musi wykonać użytkownik przed zapisaniem to wybranie punktu wewnętrz wykonanego obrysu, poprzez wcisnięcie lewego przycisku myszy w odpowiednim miejscu na obrazie — Rysunek ??.



Rysunek 5.9: Widok listy obrysów po zapisaniu obrysu manualnego

Po kliknięciu przycisku wyślij (Send) lewym przyciskiem myszy obrys zostaje zapisany i pojawia się na liście wykonanych obrysów (Available Contours). Lista zawiera jednie obrys dla oglądanego obrazu — Rysunek ??.

---

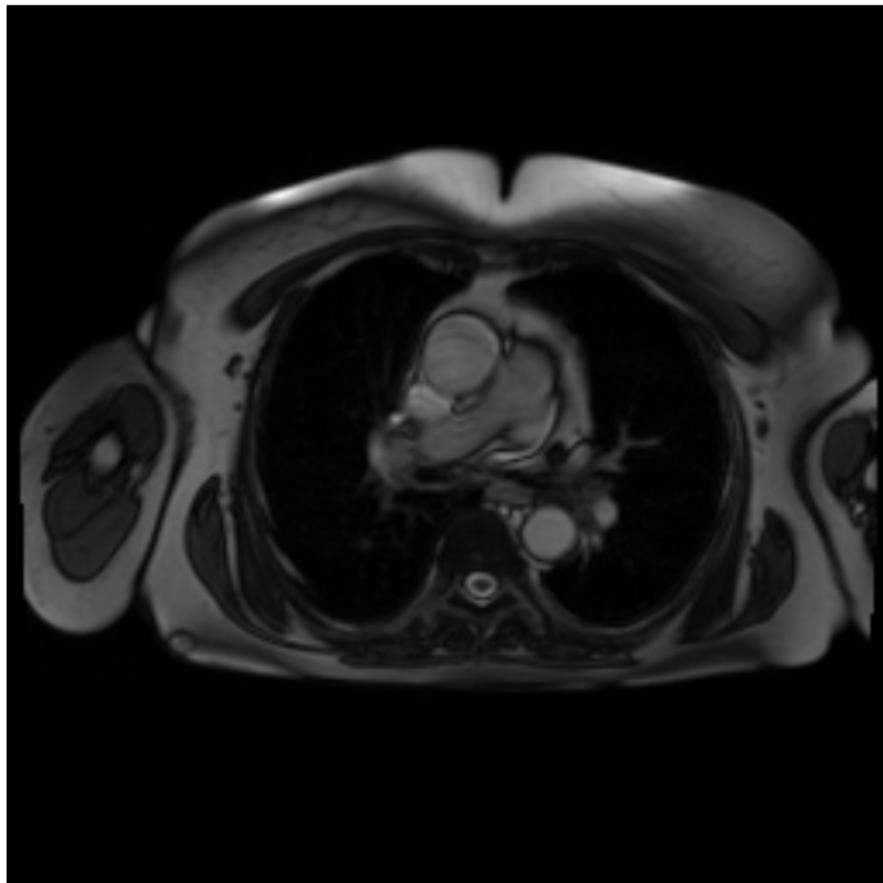
MANUAL

SEMI-AUTOMATIC

---

N/A/CMWUM^TRZUSTKA/t2\_localizer\_multi

7/15



PREVIEW CONTOUR

SAVE CONTOUR

CHOOSE COLOR

CLEAR POINTS

CLEAR GENERATED CONTOUR

Rysunek 5.10: Moduł obrysu półautomatycznego

Użytkownik może wykonywać obrys w sposób półautomatyczny. W tym celu musi wybrać kartę z obrysami półautomatycznymi (z Manual do Semi-Automatic) — Rysunek ??.

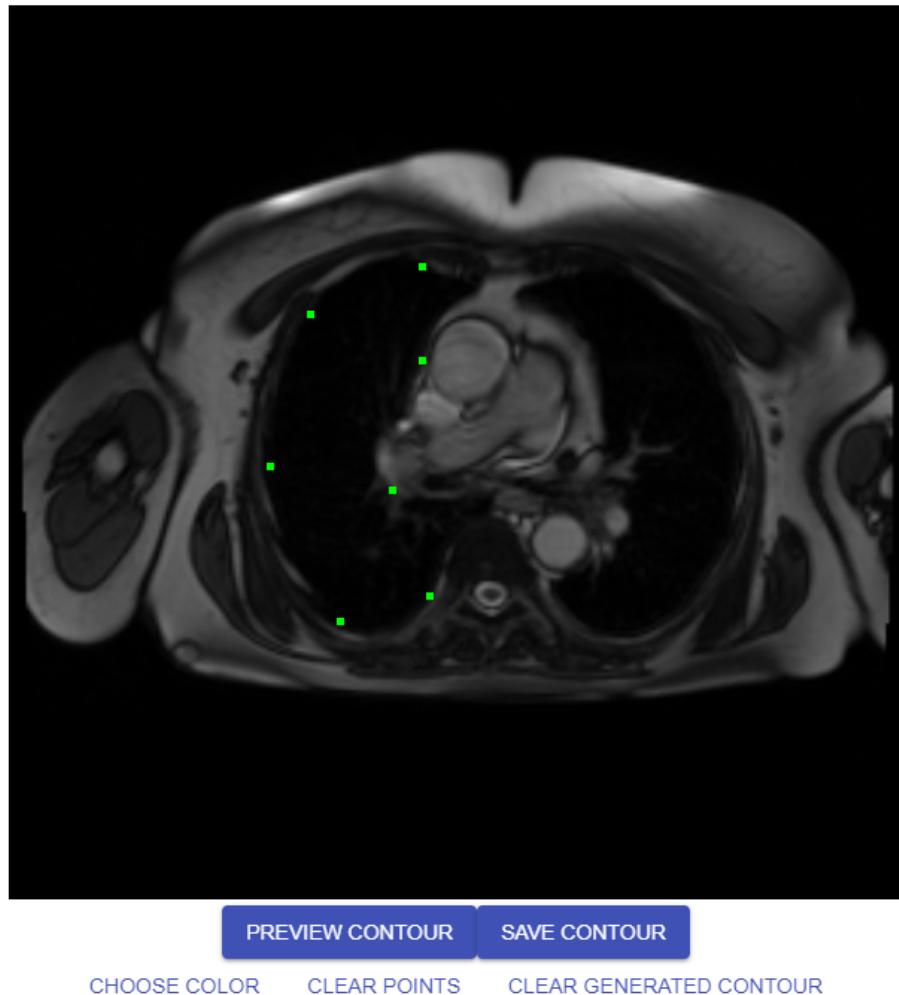
## INSTRUKCJA UŻYTKOWANIA

MANUAL

SEMI-AUTOMATIC

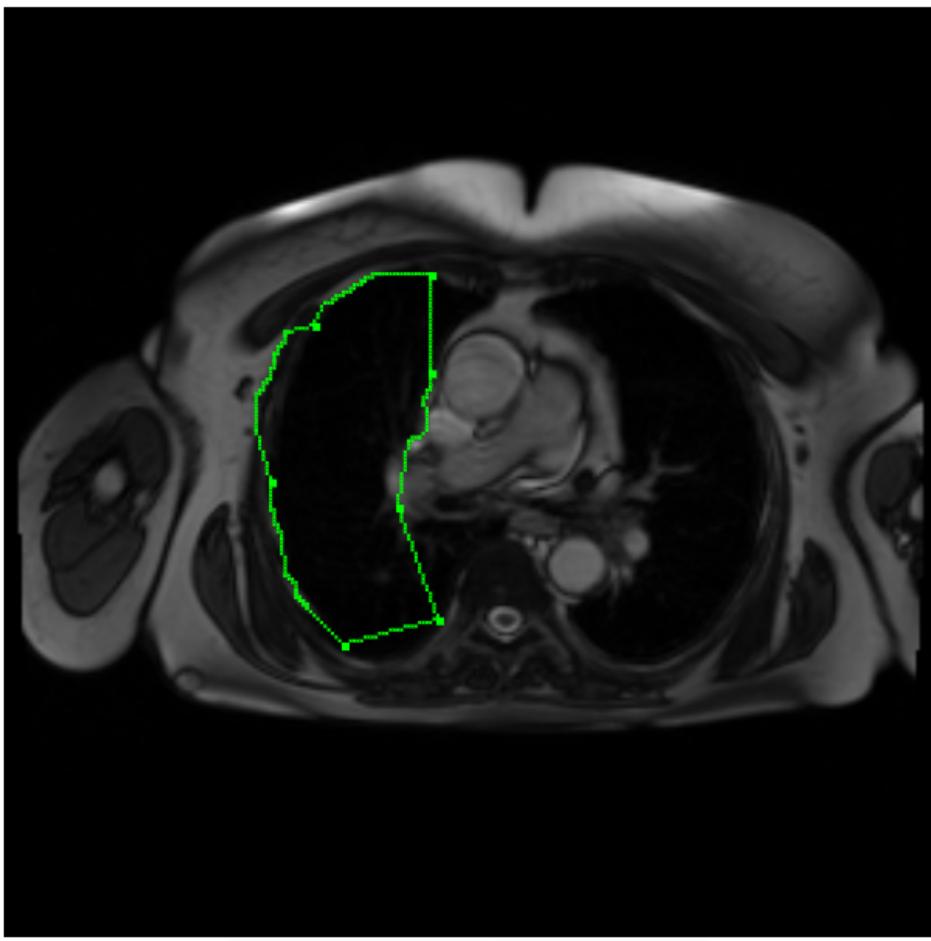
N/A/CMWUM^TRZUSTKA/t2\_localizer\_multi

7/15



Rysunek 5.11: Punkty wybrane do obrysu półautomatycznego

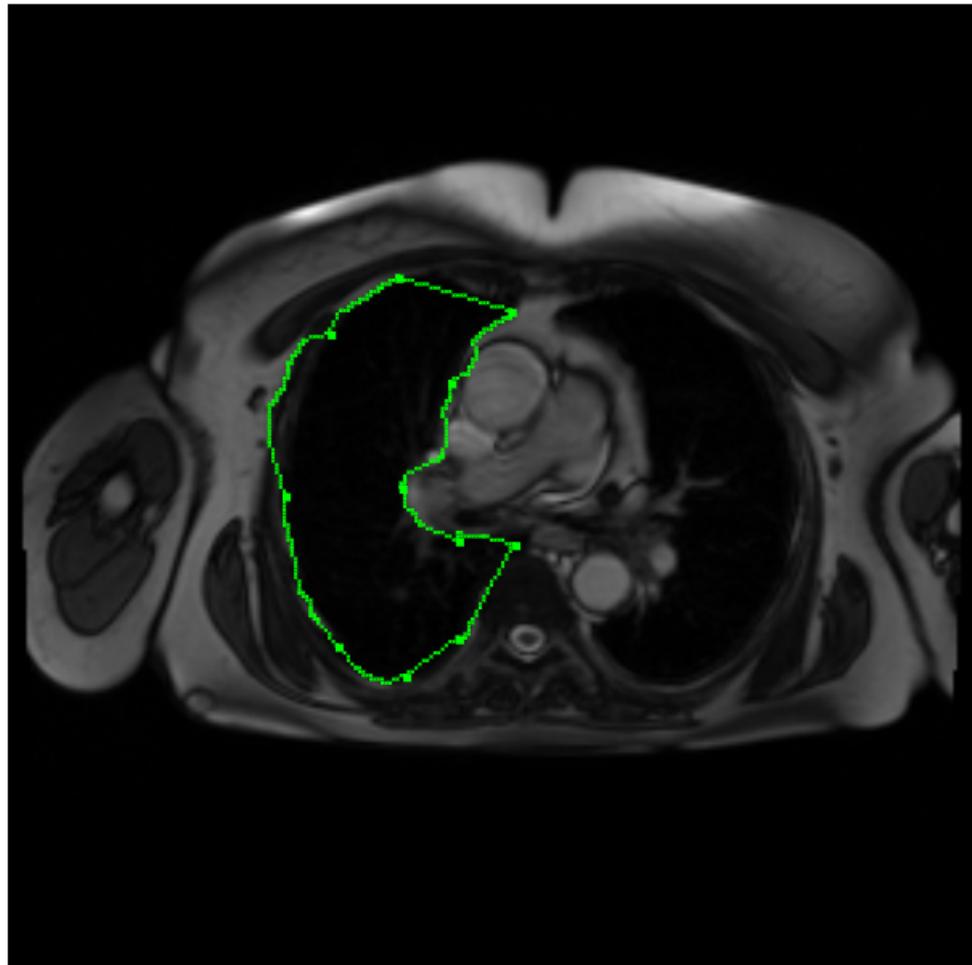
Aby wybrać punkty do obrysu półautomatycznego użytkownik powinien kliknąć w interesującego punkty lewym przyciskiem myszy. Jeśli korzysta z tabletu graficznego, powinien przycisnąć rysik do tabletu w interesujących go punktach. Punkty wprowadzane powinny być w kolejności, w której występują w obrysie, zgodnie albo przeciwnie do ruchu wskazówek zegara — Rysunek ??.



Rysunek 5.12: Podgląd obrysu półautomatycznego

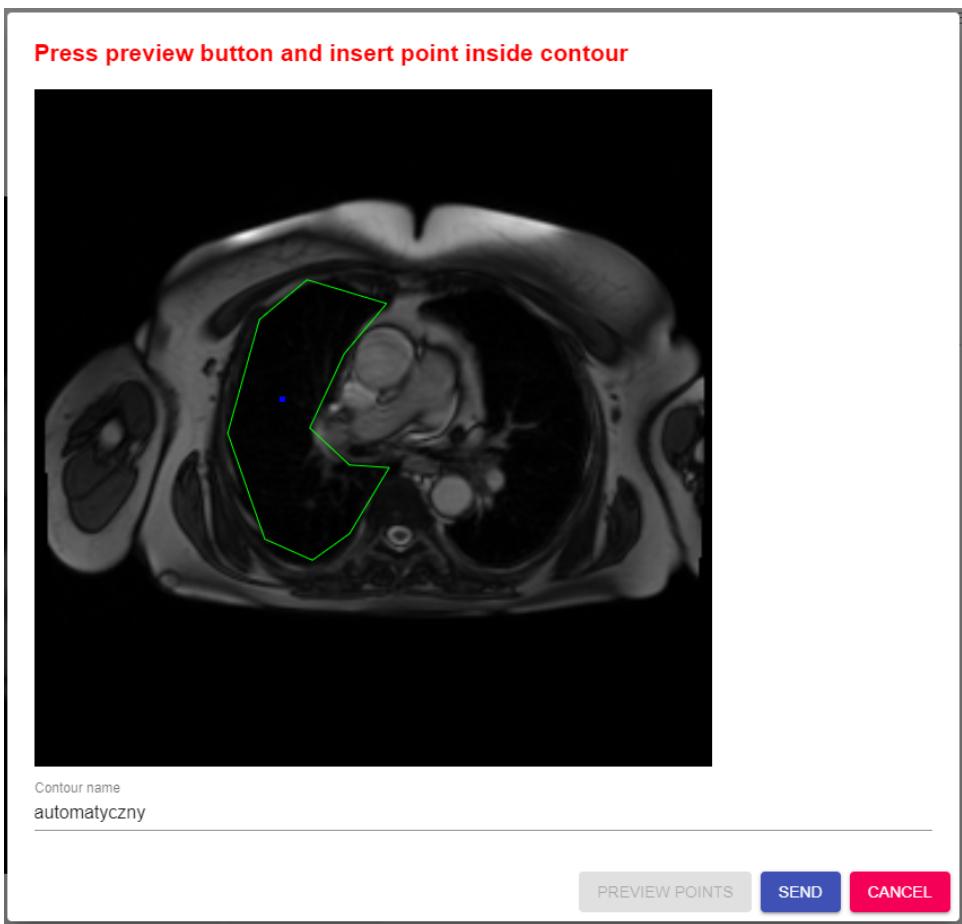
Po wciśnięciu przycisku podgląd obrysu (Preview Contour) użytkownik może zobaczyć wygenerowany przez algorytm obrys. — Rysunek ??.

## INSTRUKCJA UŻYTKOWANIA



Rysunek 5.13: Podgląd obrysu półautomatycznego po dodaniu nowych punktów

Jeśli algorytm popełnił błąd użytkownik może dodać nowe punkty klikając lewym przyciskiem myszy w interesujących go miejscach lub usunąć nietrafnie umieszczone punkty poprzez klikanie na nich lewego przycisku myszy — Rysunek ??.

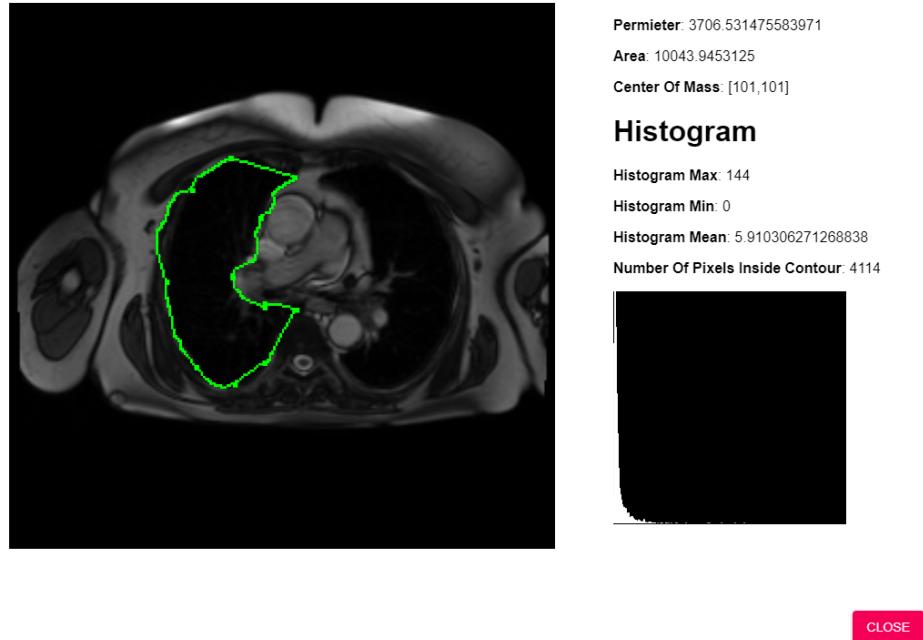


Rysunek 5.14: Widok zapisu obrysu półautomatycznego

Gdy wygenerowany obrys jest satysfakcjonujący dla użytkownika, może go zapisać klikając zapisz obrys (Save Contour). Okno zapisu jest analogiczne do okna zapisu obrysu manualnego. Jeśli na obrazie pojawią się krzyżujące krawędzie, należy wyczyścić obrys i wykonać go od początku, gdyż algorytm modyfikujący obrys dodał nowe punkty w złym miejscu lub użytkownik wprowadził punkty w złej kolejności — Rysunek ??.

## INSTRUKCJA UŻYTKOWANIA

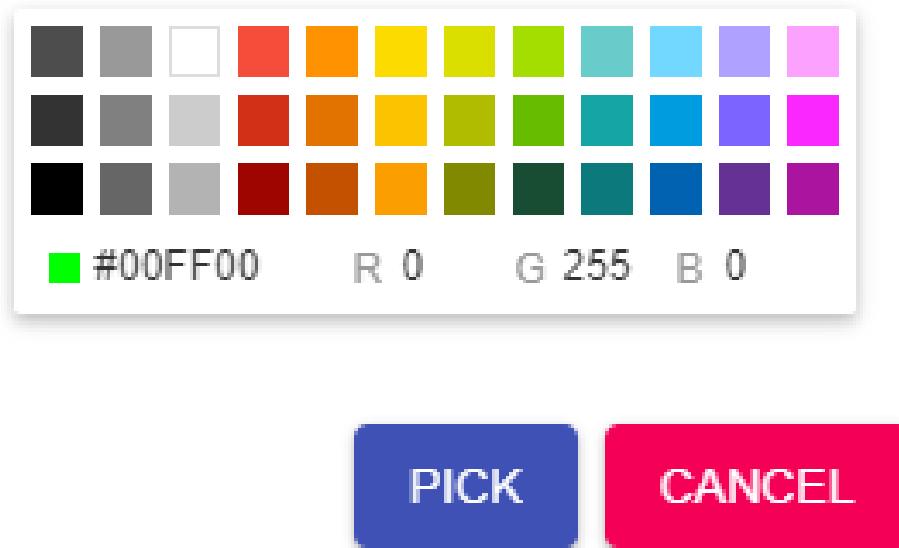
automatyczny



CLOSE

Rysunek 5.15: Podgląd statystyk obrysu

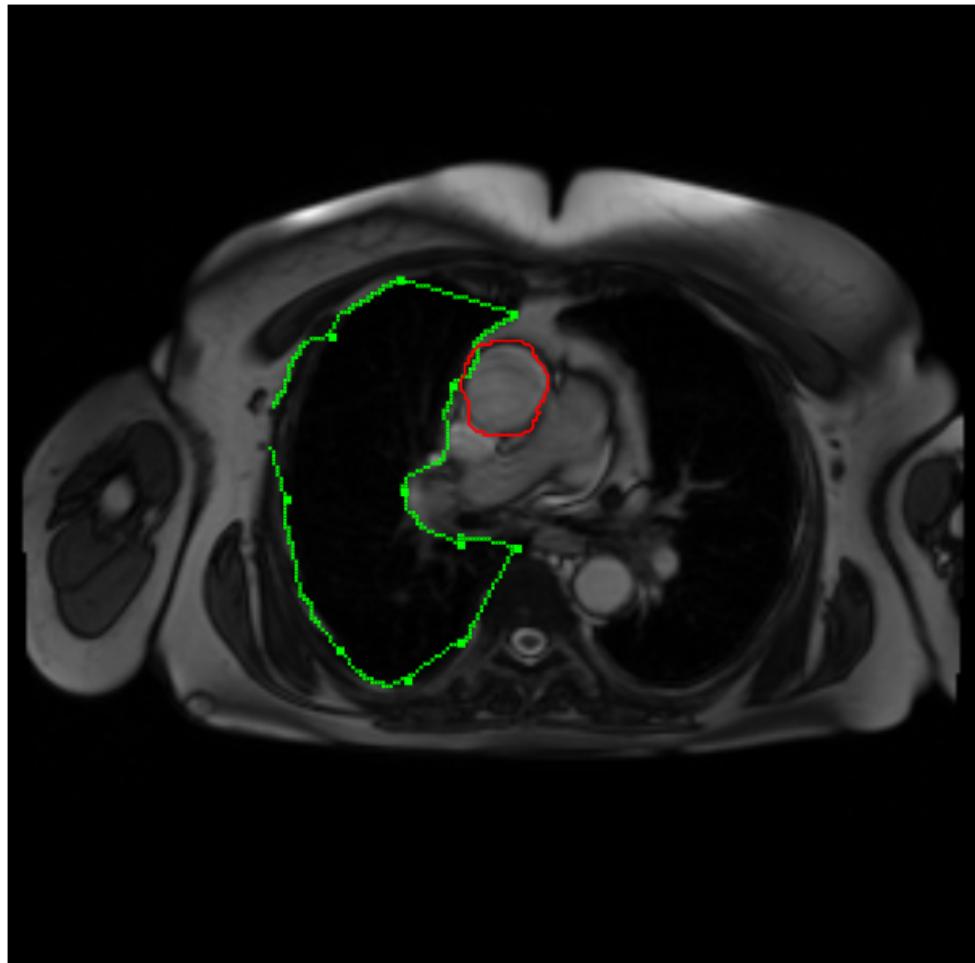
Po zapisaniu użytkownik może obejrzeć statystyki wykonanego obrysu poprzez kliknięcie lewego przycisku myszy na nazwie obrysu — Rysunek ??.



Rysunek 5.16: Widok wyboru koloru obrysu

Przed wykonaniem obrysu użytkownik może wybrać kolor wykonywanego obrysu. Po wciśnięciu lewym przyciskiem myszy na przycisk wybierz kolor (Choose Color), pojawia się okno wyboru koloru. Użytkownik może wybrać kolor z przygotowanej wcześniej palety kolorów lub wprowadzić dowolny inny korzystając z pól tekstowych, wprowadzając kod RGB wybranego koloru — Rysunek ??.

## INSTRUKCJA UŻYTKOWANIA



Rysunek 5.17: Widok wielu obrysów

W zakładce z podglądem obrysów użytkownik może wyświetlać wybrane przez niego obrysów z listy wykonanych obrysów. Użytkownik wybiera obrys, który chce zobaczyć. Obrys, który nie są aktualnie wyświetlane można dodać do obrazu klikając je lewym przyciskiem myszy na nazwie obrysu, a wyświetlane można wyłączyć również poprzez kliknięcie lewego przycisku myszy na nazwie obrysu — Rysunek ??.

**Name:** N/A

**Birthdate:** 19340608

**Sex:** F

**Institution Name:** N/A

**Referring Physician Name:** N/A

**Study Date:** 20161010

**Study Description:** CMWUM^TRZUSTKA

**Pixel Spacing:** 1.5625\1.5625

**Spacing Between Slices:** 32

**ANONYMIZE**

Rysunek 5.18: Widok szczegółów obrazu

Interfejs wyświetla również informacje dotyczące pacjenta i jego badania oraz wyświetlonego obrazu — Rysunek ??.

## INSTRUKCJA UŻYTKOWANIA

### Anonymize patient

Patient Name

N/A

Birthdate

19340608

Patient sex

F

**ANONYMIZE**

**CANCEL**

Rysunek 5.19: Anonimizacja danych pacjenta

Poprzez wciśnięcie przycisku anonimizuj (Anonymize) użytkownik może wybrać nazwę pacjenta, datę jego urodzenia oraz płeć po anonimizacji jego danych — Rysunek ??.



## Wykaz symboli i skrótów

- API — ang. Application Programming Interface — zestaw ściśle określonych reguł, poprzez które komunikują się ze sobą programy
- CRUD — ang. Create, Read, Update, Delete — utwórz, odczytaj, aktualizuj i usuń — cztery podstawowe funkcje w aplikacjach, które umożliwiają zarządzanie nią
- CSV — ang. Comma-Separated Values (w pracy użyte w kontekście formatu pliku)
- DoS — ang. Denial of Service — atak na system komputerowy mający na celu uniemożliwienie działania systemu
- DICOM — ang. Digital Imaging and Communications in Medicine — norma ujednolicająca wymianę i interpretację obrazowych danych medycznych
- HTML5 — ang. HyperText Markup Language 5 — język do tworzenia i prezentowania stron internetowych www
- HTTP — ang. Hypertext Transfer Protocol — protokół wymiany danych hipertekstowych
- HTTPS — ang. Hypertext Transfer Protocol Secure — szyfrowana wersja protokołu HTTP
- ID — ang. Identifier — unikalna nazwa przypisana do danego obiektu, pozwalająca rozróżniać obiekty w systemie
- JSON — ang. JavaScript Object Notation — lekki format wymiany danych, bazujący na podzbiorze języka JavaScript
- REST — ang. Representational State Transfer — styl architektury oprogramowania dla

systemów rozproszonych

- RGB — ang. Red, Green, Blue — model przestrzeni barw, opisanej współrzędnymi 3 barw podstawowych: czerwonej, zielonej i niebieskiej
- SDK — ang. Software Development Kit — zestaw narzędzi programistycznych niezbędny do tworzenia aplikacji korzystającej z danej biblioteki

## **Spis rysunków**

## **Spis tabel**

## **Spis zawartości załączonej płyty CD**

## Spis załączników

1. Załącznik 1 - Pseudokod generujący graf z bitmapy

```
1) MATRIX - macierz wejściowa z oznaczonymi krawędziami jako 1
2) foreach(punkt A taki, że MATRIX(A) == 1)
3) {
4)     if ( punkt A ma 1 sąsiada albo co najmniej 3 sąsiadów )
5)     {
6)         // (tzn. jest albo punktem końcowym albo węzłowym)
7)         wstaw punkt A do kolejki wierzchołków L_V
8)         while ( kolejka wierzchołków L_V niepusta )
9)         {
10)             weź wierzchołek V_1 z kolejki L_V
11)             usuń V_1 z kolejki L_V
12)             dodaj wierzchołek V_1 do grafu G
13)             foreach (punkt B sąsiadujący z V_1 )
14)             {
15)                 if ( MATRIX(B) == 1 )
16)                 {
17)                     stwórz nową krawędź E.
18)                     dodaj B do E
19)                     ustaw wierzchołek V_1 jako początek krawędzi E
20)                     dodaj krawędź E do kolejki przetwarzanych krawędzi L_E
21)                 }
22)                 while ( L_E niepusta)
23)                 {
24)                     weź krawędź E z L_E
25)                     usuń E z kolejki
26)                     stwórz kolejkę potencjalnych punktów krawędzi E, L_P
```

## SPIS ZAŁĄCZNIKÓW

```
27)           weź punkt C z E
28)           usuń C z E
29)           dodaj punkt C do kolejki L_P
30)           while ( L_P niepusta )
31)           {
32)               weź punkt D z L_P
33)               usuń D z L_P
34)               K = liczba sąsiadów D
35)               if ( K == 0 lub K > 1 )
36)               {
37)                   stwórz wierzchołek V_2, który znajduje się w D
38)                   do listy krawędzi wierzchołka V_2 dodaj E
39)                   do listy krawędzi wierzchołka V_1 dodaj E
40)                   ustaw wierzchołek V_2 jako koniec krawędzi E
41)                   dodaj krawędź E do grafu G
42)                   dodaj wierzchołek V_2 do kolejki L_V
43)               }
44)               if ( K == 1 )
45)               {
46)                   dodaj punkt D do E
47)                   foreach ( punkt F sąsiadujący z D )
48)                   {
49)                       if ( MATRIX(F) == 1 )
50)                           {
51)                               dodaj F do L_P
52)                           }
53)                       }
54)                   }
55)                   MATRIX(D) = 1
56)               }
57)           }
58)       }
59)   }
60) }
```

61) }

62) Zwróć graf G